

**UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE SISTEMAS DE INFORMAÇÃO**

Guia de testes de segurança para aplicações web

Antonio Marco da Costa Taha

**Florianópolis - SC
2017 / 2**

UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE SISTEMAS DE INFORMAÇÃO

Guia de testes de segurança para aplicações web

Antonio Marco da Costa Taha

Trabalho de conclusão de
curso apresentado como
parte dos requisitos para
obtenção do grau de
Bacharel em Sistemas de
Informação

Florianópolis - SC
2017 / 2

Antonio Marco da Costa Taha

Guia de testes de segurança para aplicações web

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Sistemas de Informação.

Prof. Carla Merkle Westphall

Orientadora

Banca examinadora:

Prof. Carlos Becker Westphall

Prof. João Bosco Manguiera Sobral

Lucas Marcus Bodnar

RESUMO

As aplicações web são cada vez mais frequentes no cotidiano das pessoas e facilitam a realização de tarefas, como operações bancárias através de sistemas na Internet. Uma grande parcela das aplicações web armazenam e processam dados sensíveis fornecidos por seus usuários, portanto precisam cuidar da segurança desses dados. Este trabalho tem por objetivo elaborar um guia de testes de segurança para aplicações web, baseado na metodologia da OWASP. Os testes abordados pelo guia apresentam através de exemplos práticos como detectar e explorar vulnerabilidades em sistemas web e utilizam ferramentas de segurança no seu processo. Com o auxílio deste guia, através de um roteiro simples e intuitivo, alunos de graduação, profissionais com pouco conhecimento em segurança de sistemas e interessados, podem aprender como detectar vulnerabilidades em suas aplicações e como elas surgem.

Palavras-chave: Aplicações Web, Guia de testes, OWASP, Testes de Segurança, Vulnerabilidades.

LISTA DE FIGURAS

Figura 01 - Modelo cliente-servidor de 3 camadas	10
Figura 02 - Modelo de formulário de cadastro.....	25
Figura 03 - Formulário padrão de autenticação	28
Figura 04 - Consulta SQL de usuário	29
Figura 05 - Consulta SQL com <i>string</i> de ataque	29
Figura 06 - Tela de autenticação do Google	32
Figura 07 - Credenciais de autenticação.....	32
Figura 08 - Formulário para visualização dos dados de usuário	36
Figura 09 - Campo HTML de e-mail.....	36
Figura 10 - Campo HTML com código injetado.....	37
Figura 11 - Alerta gerado por injeção de código	37
Figura 12 - Busca de endereço do usuário	38
Figura 13 - Url para ataque de XSS Refletido	38
Figura 14 - Alerta de XSS Refletido	38
Figura 15 - Mutillidae.....	41
Figura 16 - WordPress	42
Figura 17 - Kali Linux no virtualbox	44
Figura 18 - OWASP Zap	45
Figura 19 - Diagrama de execução do desenvolvimento	48
Figura 20 - Imagens de download Kali Linux	49
Figura 21 - Importação de máquina virtual.....	50
Figura 22 - Máquinas virtuais importadas	50
Figura 23 - Redes Exclusivas de Hospedeiro	51
Figura 24 - Endereço ip da placa virtual.....	52
Figura 25 - Servidor DHCP	52
Figura 26 - Adaptador 1 da máquina Kali Linux	53
Figura 27 - Configuração da placa NAT	54
Figura 28 - Adaptador 1 da máquina Owasp Broken	54
Figura 29 - Interfaces de rede da Kali Linux.....	55
Figura 30 - Placas de rede na interface gráfica da Kali Linux	56

Figura 31 - Placas de rede ativas e com Profile 1	57
Figura 32 - Inicialização da máquina virtual Owasp Broken.....	58
Figura 33 - Abrindo a ferramenta Owasp Zap	59
Figura 34 - Básico da Owasp Zap	60
Figura 35 - Configuração do <i>proxy</i> na Owasp Zap.....	62
Figura 36 - Certificado SSL	63
Figura 37 - Abrir navegador Firefox	64
Figura 38 – <i>Preferences</i>	64
Figura 39 - Configuração do <i>proxy</i> no navegador	65
Figura 40 - Aplicações da máquina Owasp Broken	68
Figura 41 - <i>Link</i> para acesso a página de registro	69
Figura 42 - Página de registro para teste	70
Figura 43 - Registro da requisição na Owasp Zap	71
Figura 44 - Execução de varredura ativa	71
Figura 45 - Resultado da varredura ativa	72
Figura 46 - Exploração de vulnerabilidade de XSS Refletido	73
Figura 47 - Blog para teste na Mutillidae.....	76
Figura 48 - Campo de comentário em blog	76
Figura 49 - Varredura ativa ao salvar comentário	77
Figura 50 - Campo vulnerável a XSS persistido.....	77
Figura 51 - Código HTML da página de comentários do blog	78
Figura 52 - Salvar código malicioso como comentário	79
Figura 53 - Exploração de vulnerabilidade de XSS Persistido	79
Figura 54 - Formulário de busca de dados dos usuários na Mutillidae	82
Figura 55 - Busca de usuário teste	83
Figura 56 - Alertas de injeção SQL	84
Figura 57 - Formulário do teste de injeção SQL.....	84
Figura 58 - Erro do banco de dados.....	85
Figura 59 - <i>String</i> de ataque de injeção SQL	86
Figura 60 - Resultado da injeção de código SQL.....	87
Figura 61 - Tela inicial da aplicação WackoPicko	89
Figura 62 - Formulário de <i>login</i> WackoPicko.....	90
Figura 63 - Resultado de varredura na WackoPicko	90
Figura 64 - Erro na consulta SQL de <i>login</i> WackoPicko.....	91

Figura 65 - Tentativa de <i>login</i> na aplicação WackoPicko	91
Figura 66 - Esquema de autenticação burlado por injeção SQL	91
Figura 67 - Caminho para acessar página de teste	93
Figura 68 - Registrar usuário na Mutillidae.....	93
Figura 69 - Campos de usuário preenchidos	94
Figura 70 - Autenticação do usuário bobtester.....	94
Figura 71 - Inspeção da página da aplicação Mutillidae.....	95
Figura 72 - <i>Cookies</i> da aplicação Mutillidae.....	96
Figura 73 - Usuário mariatester autenticado na Mutillidae	97
Figura 74 - Alteração do valor do <i>cookie</i>	97
Figura 75 - Novo usuário autenticado na Mutillidae	98
Figura 76 - Usuário administrador autenticado	98

LISTA DE ABREVIATURAS E SIGLAS

CERN - *European Organization for Nuclear Research*

CERT.br - Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança no Brasil

CID - Confidencialidade, integridade e disponibilidade

DDL - *Data Definition Language*

DDOS - *Distributed Denial of Service*

DHCP - *Dynamic Host Configuration Protocol*

HTML - *Hypertext Markup Language*

HTTP - *Hypertext Transfer Protocol*

IP - *Internet Protocol*

NAT - *Network address translation*

OWASP - *Open Web Application Security Project*

SQL - *Structured Query Language*

SSL - *Secure Socket Layer*

URL - *Uniform Resource Locator*

VPN - *Virtual private network*

W3C - *World Wide Web Consortium*

XSS - *Cross-site scripting*

SUMÁRIO

1 INTRODUÇÃO	1
1.1 OBJETIVOS.....	4
1.1.1 Objetivo geral	4
1.1.2 Objetivos específicos	5
1.2 ESTRUTURA DO TEXTO	5
2 FUNDAMENTAÇÃO TEÓRICA.....	6
2.1 APLICAÇÕES WEB.....	6
2.1.2 Surgimento da Web	6
2.1.3 Arquitetura Cliente-Servidor	9
2.1.4 Aplicações.....	10
2.2 CONCEITOS DE SEGURANÇA.....	11
2.2.1 Confidencialidade, Integridade e Disponibilidade.....	12
2.2.2 Vulnerabilidade e Ameaça	15
2.2.3 Ataques	16
2.3 OPEN WEB APPLICATION SECURITY PROJECT	18
2.3.1 Técnicas de Teste	19
2.3.1.1 Revisão e Inspeção de Manuais.....	19
2.3.1.2 Modelagem de Ameaças	19
2.3.1.3 Revisão do Código Fonte	20
2.3.1.4 Testes de Intrusão	20

2.3.2 Tipos de Teste	22
2.3.2.1 Teste de Autenticação	23
2.3.2.2 Teste de Validação de Entrada	23
2.4 SEGURANÇA DE APLICAÇÕES WEB	26
2.4.1 Injeção de Código	27
2.4.2 Quebra de autenticação e Gerenciamento de Sessão.....	31
2.4.3 <i>Cross-Site Scripting</i>	35
3 FERRAMENTAS	39
3.1 OWASP <i>BROKEN WEB APPLICATIONS PROJECT</i>	40
3.2 KALI LINUX	43
3.3 OWASP <i>ZED ATTACK PROXY</i>	44
4 GUIA PARA TESTES DE INTRUSÃO	45
4.1 PROPOSTA.....	46
4.2 CONFIGURAÇÃO DO AMBIENTE	48
4.3 EXECUÇÃO DOS TESTES DE VALIDAÇÃO DE ENTRADA.....	65
4.3.1 Teste de <i>Cross-Site Scripting</i> Refletido	65
4.3.2 Teste de <i>Cross-Site Scripting</i> Persistido	73
4.3.3 Teste de SQL <i>Injection</i>	80
4.4 EXECUÇÃO DOS TESTES DE AUTENTICAÇÃO	87
4.4.1 Teste para burlar esquema de autenticação com injeção SQL.....	87
4.4.2 Teste para burlar esquema de autenticação via <i>cookie</i>	92

5 CONCLUSÃO	99
REFERÊNCIAS BIBLIOGRÁFICAS.....	100

1 INTRODUÇÃO

Tarefas diárias que antes precisavam de deslocamento físico das pessoas são cada vez mais realizadas através da Internet, desde uma simples compra de produtos de um supermercado até grandes transações bancárias. É impressionante como a Internet possibilitou a redução das barreiras de comunicação global, revolucionou a forma como o comércio é realizado, fornece acesso e distribui o conhecimento à sociedade, além de facilitar a vida de milhares de pessoas diariamente.

Desde o desenvolvimento da World Wide Web em 1990, por Tim Bernes-Lee, o uso da Internet tem se popularizado. A World Wide Web e sua ideia de distribuição e acesso a informação através de browser, evoluiu e nos permite hoje ter acesso a muitos recursos online.

Há diversos sites e aplicações que fornecem serviços a usuários na web. Hoje existem serviços de *streaming* através dos quais usuários podem assistir a filmes e séries, seja no computador, dispositivo móvel ou televisão. Existem aplicações que permitem a troca de mensagens, a realização de conversas por áudio e vídeo. Sistemas de Internet banking, que são sistemas de acesso à contas bancárias através da Internet, permitindo a realização de operações financeiras por usuários autorizados, seja por meio de autenticação com usuário e senha, certificado digital ou identificação biométrica. Também há redes sociais e serviços de busca nos quais milhares de informações são compartilhadas diariamente com o mundo todo. E o que todas essas aplicações possuem em comum?

Geralmente esses serviços solicitam aos seus usuários a realização de um cadastro de suas informações pessoais. Alguns cobram pelo acesso aos seus recursos e outros dizem garantir o sigilo de conversas através do uso de criptografia ponto-a-ponto. Como os desenvolvedores dessas e das inúmeras aplicações que existem na web podem garantir a confiabilidade de seus sistemas?

Se um serviço é pago para ser acessado então ele deve estar disponível ao seu usuário quando o mesmo tiver direito ao acesso. Se a aplicação deixa claro em seu contrato de uso que os dados são criptografados, ninguém mais além de quem está autorizado deve conseguir decifrar esses dados. Se um e-mail é enviado, o usuário deseja que seu conteúdo seja recebido pelo destinatário integralmente.

Conforme Peixinho e Fonseca (2013), um sistema, seja web ou desktop, pode apresentar diversos componentes, que funcionam em conjunto, como servidores físicos, o sistema operacional que estará em execução nos servidores, a própria aplicação desenvolvida e outros recursos como sistemas de bancos de dados. Torna-se inviável e foge da responsabilidade de um desenvolvedor de software cuidar da segurança de cada um desses componentes, cabendo a um ou mais profissionais de segurança a preocupação com a segurança de cada um desses recursos empregados.

Ainda segundo Peixinho e Fonseca (2013), os desenvolvedores de software possuem cada vez menos tempo para a entrega do produto final ao cliente, que tem se tornado cada vez mais exigente, e não possuem o conhecimento técnico necessário para garantir a segurança de uma aplicação.

Stallings (2008) comenta que devido ao aumento na complexidade das aplicações e do acesso à Internet, bem como o uso de recursos que dependem

cada vez mais de conexão com a rede, é necessário cada vez mais profissionais com conhecimento técnico em segurança e tecnologias capazes de agir contra as ameaças aos sistemas.

“Ao longo do tempo, os ataques na Internet e em sistemas conectados à Internet se tornaram mais sofisticados, enquanto a habilidade e o conhecimento exigidos para montar um ataque diminuíram.” (STALLINGS, 2008, p. 5).

Existem padrões de projetos e técnicas de programação que permitem aos desenvolvedores produzirem código mais seguro, evitando vulnerabilidades nas aplicações conectadas à Internet, porém, devido ao curto prazo de entrega exigido por clientes e também pela falta de conhecimento em segurança da informação, muitas aplicações web entram em produção contendo vulnerabilidades que poderiam ser evitadas.

Peixinho e Fonseca (2013, p. 182) definem vulnerabilidade como sendo “uma brecha em um sistema computacional” e dizem ainda que “Quando tratamos de programas (software), essas vulnerabilidades são muitas vezes chamadas de bugs.”

Brechas podem ser exploradas por atacantes para roubar informações sigilosas dos usuários das aplicações, para deixar sistemas indisponíveis como é o caso de servidores e aplicações financeiras, causando grande prejuízo financeiro às instituições que usam ou são proprietárias de tais serviços e ainda problemas mais recentes como o ransomware, que é o sequestro de máquinas pessoais, servidores e sistemas inteiros, cuja devolução do acesso é realizada mediante pagamento em moedas virtuais como Bitcoin, que não são rastreadas, conforme diz a Cartilha de Segurança para Internet do CERT.br (2012).

Com o objetivo de detectar tais vulnerabilidades nos sistemas, impedir o acesso não autorizado de pessoas à dados sigilosos e que estes dados não sejam adulterados, além de evitar indisponibilidade das aplicações devido a falhas de segurança, devem ser realizadas auditorias de segurança. Peixinho e Fonseca (2013) definem que auditoria trata-se de um processo de comparação de algo contra um padrão estabelecido e destacam que em sistemas de informação, vários parâmetros podem ser utilizados para auditar um sistema, como a política de segurança organizacional ou se um sistema está atualizado conforme as novas correções de segurança disponíveis no mercado.

Segundo Menezes, Cardoso e Rocha (2015), um dos meios de se realizar a auditoria de segurança em sistemas de informação é através da realização de testes de penetração. Esses testes simulam eventuais ataques que possam acontecer ao sistema e permitem a descoberta de falhas e aberturas no sistema, que podem comprometer as propriedades de confidencialidade, integridade e disponibilidade de uma aplicação.

1.1 OBJETIVOS

1.1.1 Objetivo geral

Este trabalho tem como objetivo geral a elaboração de um guia de testes de intrusão voltado às aplicações web, tendo como base a metodologia proposta pela Open Web Application Security Project (OWASP) em seu documento Testing Guide, atualmente na versão 4.0 (OWASP, 2014). O objetivo deste guia é permitir que profissionais sem conhecimentos profundos em segurança possam testar suas

aplicações web e detectar as principais vulnerabilidades descritas pela OWASP, seguindo um roteiro simples e intuitivo.

1.1.2 Objetivos específicos

Como objetivos específicos deste trabalho temos:

- Desenvolver um guia de testes de intrusão em aplicações web;
- Realizar testes de penetração utilizando o guia gerado por este trabalho em ambientes controlados;
- Demonstrar os experimentos realizados com o guia, explicando o funcionamento das ferramentas utilizadas para a realização dos testes;
- Apresentar quais as vulnerabilidades foram encontradas nos ambientes através da execução dos procedimentos descritos no guia.

1.2 ESTRUTURA DO TEXTO

Este trabalho está dividido nos capítulos de Introdução, Fundamentação Teórica, Ferramentas, Guia para Testes de Intrusão e Conclusão.

No capítulo de Fundamentação Teórica são levantados e explicados todos os conceitos envolvidos para a execução de testes de segurança em aplicações web. São comentados os principais conceitos de segurança, diferenciando vulnerabilidade de ameaça e explicando ataques a segurança de sistemas. No capítulo também são explicadas as vulnerabilidades que serão testadas com o apoio do guia e os testes que serão realizados.

No capítulo de Ferramentas são apresentadas duas máquinas virtuais que serão utilizadas para a execução dos testes de intrusão e a ferramenta OWASP Zap, utilizada para varredura de vulnerabilidades em aplicações web.

O capítulo Guia para Testes de Intrusão apresenta toda a configuração do ambiente, necessária para a execução dos testes apresentados no guia. São demonstrados minuciosamente como executar cada teste de intrusão através de exemplos práticos.

No capítulo de conclusão são levantados quais objetivos foram alcançados com o uso do guia desenvolvido e comentando a respeito de trabalhos futuros que possam ser realizados com o apoio do guia.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 APLICAÇÕES WEB

2.1.2 Surgimento da Web

Segundo Aquino (2006), em 1989, o Centro Europeu de Pesquisa Nuclear (CERN), enfrentava problemas para realizar a comunicação entre seus pesquisadores, que eram lotados em diferentes projetos desenvolvidos não só na base do instituto, localizada na Suíça. Para trocar informações a respeito das pesquisas realizadas, os cientistas do CERN ainda utilizavam documentos em papel, gerando cada vez mais atrasos na troca de informações entre as equipes de pesquisadores e o risco da perda de conhecimento. Para solucionar esse problema o engenheiro Tim Berners-Lee criou em 1990 a World Wide Web.

“A World Wide Web é uma iniciativa para recuperação de informação hipermídia em longa distância que tem como objetivo prover acesso a um amplo universo de documentos” conforme o próprio site do CERN (2017). Trata-se da forma de distribuir e dar acesso a documentos hipermídia através da Internet, na qual um usuário solicita através de um browser um documento ou página localizada em determinado endereço que referencia um servidor web que estará provendo tal conteúdo. Além disso Tim Berners-Lee desenvolveu uma linguagem de marcação de texto, conhecida por *HyperText Markup Language* (HTML), que deveria ser utilizada para a construção dos documentos que seriam compartilhados através da web.

O crescimento gigantesco da internet aconteceu devido ao uso da web e sua forma de distribuição de conteúdo, que ultrapassou a tarefa de compartilhar conhecimento entre cientistas de um centro de pesquisas e permitiu a troca de informações entre pessoas do mundo inteiro.

Para entender o funcionamento básico da web é necessário conceituar alguns termos básicos como HTML, URL e links. A maioria dos documentos presentes na web são conhecidos também como páginas web, que apresentam conteúdo do tipo textual, imagens, vídeos e outros. Essas páginas podem ser localizadas por meio de um identificador. Oliveira (2012) descreve que cada página web apresenta um identificador único conhecido por URL (*Uniform Resource Locator*). A URL é uma sequência de caracteres, que segue um determinado esquema e conjunto de regras de construção, que tem por objetivo identificar e permitir a localização de um recurso disponível na web, conforme apresentado na RFC 1738 (BERNERS-LEE; MASINTER; MCCAILL, 1994). Pode-se dizer que o

esqueleto dessas páginas é o HTML, que vai definir como o conteúdo visualizado será estruturado e como será agrupado em seções diferentes. O HTML é uma linguagem de marcação que descreve ao navegador como uma página web deve ser exibida ao usuário, marcando textos e outras informações com elementos conhecidos por *tags*. Esses elementos podem determinar se um conteúdo é uma imagem, determinar que certo texto esteja em negrito e conectar documentos entre si. As páginas web podem estar conectadas entre si através de *links*, que são elementos que indicam ao navegador que ele deve requisitar determinado conteúdo numa URL específica.

Sendo assim, uma pessoa com acesso à Internet pode se conectar a web e visualizar uma página através de um navegador, apenas digitando a URL, e o navegador irá encaminhar a requisição ao servidor, que provê o conteúdo neste endereço, e que retornará como resposta uma página web ou determinado tipo de conteúdo, que serão mostrados no próprio navegador.

Atualmente a web não trata mais somente de acesso a páginas estáticas conectadas através de *links*, (UTO, 2013). Os desenvolvedores viram na Web uma grande oportunidade para começar a criar aplicações distribuídas e essas aplicações permitem que inúmeros usuários possam acessar seus recursos através de um navegador, sem precisar instalar o software localmente. De acordo com os dados gerados pelo usuário, uma aplicação pode fornecer diferentes respostas e dinamicamente alterar seu conteúdo em resposta às requisições ao servidor de aplicações que deverá estar online. Nesses casos é necessário a conexão com a Internet, apesar de que já existem aplicações que permitem você trabalhar

localmente sem conexão e quando estiver online é realizada uma sincronia dos dados.

2.1.3 Arquitetura Cliente-Servidor

Ao solicitar páginas ou serviços na web, um usuário está utilizando o modelo cliente-servidor. Segundo Held (2000) essa arquitetura define basicamente que um ou mais computadores, conectados numa rede, solicitem a um ou mais servidores na mesma rede ou na Internet, conteúdo e serviços, que estes não possuem acesso ou que não podem estar disponíveis no lado cliente sem antes passar por um conjunto de medidas de segurança para fornecimento do serviço ou por questões de capacidade de processamento de dados.

Atualmente os sistemas na web já trabalham com um modelo conhecido por arquitetura de 3 camadas. Held (2000) descreve que essa arquitetura apresenta “um servidor de aplicação que é a peça central do novo modelo. Ele pode ser descrito como um middleware, localizado entre clientes e fontes de dados ou sistemas legados, que gerencia e processa aplicativos de três camadas”.

Nos sistemas em 3 camadas, figura 1, o cliente solicita a maioria dos recursos ao servidor de aplicações pois existem muitos recursos que trabalham com grande processamento e volume de dados, com cálculos complexos, *upload* de conteúdo multimídia e que demandam alto poder de processamento nem sempre disponível nas máquinas cliente.

O modelo Web adotado nessa arquitetura, isola o acesso aos dados do lado cliente, tentando evitar questões de segurança, por isso temos geralmente o acesso

a um banco de dados somente através de requisições ao servidor, que deverá dar ou não permissão de acordo com os privilégios do usuário.

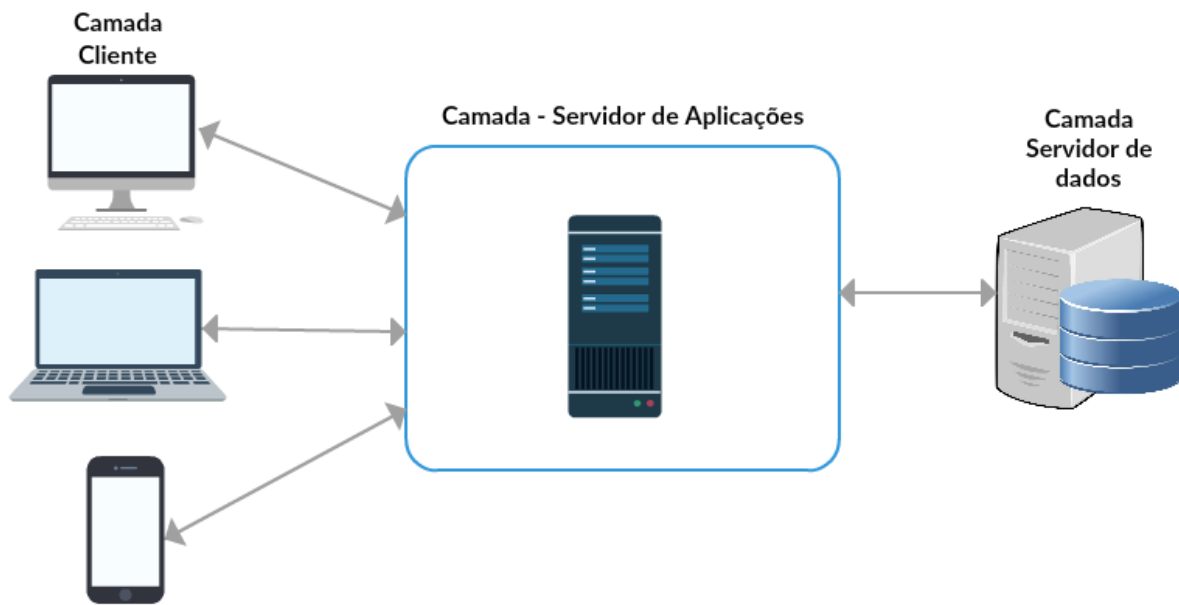


Figura 01 - Modelo cliente-servidor de 3 camadas

2.1.4 Aplicações

Para (KAPPEL, 2004) “uma aplicação Web é um sistema de software baseado em tecnologias e padrões do World Wide Web Consortium (W3C) que provê recursos específicos de Web, como conteúdo e serviços através de uma interface de usuário, o Web browser”.

(DI LUCCA e FASOLINO, 2006) afirmam que uma aplicação web pode ser considerada como um sistema distribuído, apresentando uma arquitetura multicamadas, com acesso concorrente por muitos usuários em diversos lugares do mundo, compatível com ambientes de execução heterogêneos. Ainda segundo os autores, uma aplicação web pode ser componentizada, isto é, cada componente

pode apresentar tecnologias diferentes, como a linguagem de programação utilizada em sua construção, e ter a habilidade de gerar conteúdo dinâmico de acordo com a interação do usuário, a entrada de dados e o estado do servidor.

Essas aplicações são sistemas mais complexos do que páginas estáticas na web. A interação com o usuário se torna mais frequente, existe um enorme número de requisitos funcionais que permitem aos usuários gerar dados que na maioria das vezes são persistidos em banco de dados e que são utilizados para definir conteúdo dinâmico a ser apresentado pela própria aplicação. Essas aplicações geralmente utilizam o modelo de 3 camadas, onde na camada de cliente ou apresentação o usuário através da interface da aplicação faz requisições à camada de lógica de negócio, especificamente ao servidor de aplicações, que pode solicitar ou enviar informações à camada de dados, e também responder às requisições, gerando conteúdo dinâmico a ser apresentado na camada cliente.

2.2 CONCEITOS DE SEGURANÇA

O surgimento da Web e o rápido crescimento de computadores conectados às redes, tem feito as pessoas compartilharem cada vez mais informações na Internet. Hoje existem inúmeros sistemas complexos e que processam dados sensíveis de milhares de pessoas. Pode-se citar como exemplo o *Internet banking*, um sistema de gerenciamento bancário através do qual as pessoas podem realizar tarefas que antes eram realizadas somente nas agências bancárias. Tendo esse sistema como exemplo, pode-se tentar imaginar que tipos de dados o usuário pode estar trocando com o servidor que mantém essa aplicação. *Login* e senha são

exemplos de informações que o usuário precisa preencher para conseguir acessar o sistema. Se um usuário for realizar uma transferência de recursos, ele precisará novamente digitar a senha para confirmar a transação e possivelmente informar uma chave de segurança que o mesmo recebeu da instituição, ou ter acesso a certificados digitais que comprovam sua autenticidade.

Num sistema de e-mail de uma organização é possível que ocorra a troca de informações confidenciais e importantes. Esses e outros exemplos, demonstram que nós compartilhamos e guardamos muitas informações, seja através da Internet ou em redes locais, e que precisam de um certo nível de segurança para evitar que terceiros tenham acesso indevido, que usem nossos dados ou até mesmo se façam passar por nós.

“Segurança da informação é a proteção da informação de vários tipos de ameaças para garantir a continuidade do negócio, minimizar o risco ao negócio, maximizar o retorno sobre os investimentos e as oportunidades de negócio.” (ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS, 2005, p. 9).

2.2.1 Confidencialidade, Integridade e Disponibilidade

A confidencialidade, a integridade e a disponibilidade (CID), compõem as três propriedades fundamentais de segurança, que são conhecidos como seus três pilares de sustentação.

Recursos computacionais, documentos e informações que devem estar protegidos e em segredo só podem ser acessados por entidades com autorização para isso. Segundo Pfleeger e Pfleeger (2007), a confidencialidade é a propriedade

que garante que esses recursos só possam ser acessados por quem realmente tem autorização, e se não a tiver, não deve ser possível visualizar, acessar, distribuir ou imprimir algo.

Os dados de um prontuário médico eletrônico só podem ser visualizados pelos profissionais da saúde que tenham permissão e acesso ao prontuário e pelo próprio paciente a quem se destina o prontuário, e caso alguém não autorizado tenha acesso a esse documento, inclusive físico, perde-se a propriedade de confidencialidade.

Acesso indevido à e-mails, mensagens criptografadas, contas de inúmeros serviços, roubo de dados de transações eletrônicas, são exemplos de problemas ligados à confidencialidade. Bishop (2005) afirma que existem mecanismos para o controle de acesso aos dados, como a criptografia, que geralmente através de inúmeras operações matemáticas e de controle, torna os dados incompreensíveis, criptografados.

Os algoritmos criptográficos mais atuais costumam fazer uso de chaves, que são parâmetros passados ao algoritmo para que este cifre os dados e que alguém não autorizado não possa decifrá-los sem conhecer a chave. Isso gera outro problema, deve-se garantir a confidencialidade da chave.

A propriedade de integridade tem como objetivo garantir se dada informação ou recurso é confiável ou não, normalmente trata da integridade dos dados, ou seja, do conteúdo da informação, e da origem desses dados (BISHOP, 2005). Uma informação é íntegra e confiável se ela não sofreu nenhuma adulteração e se for possível confirmar ou houver garantias da integridade da sua origem. Pode-se citar como exemplo o caso de um usuário realizando o *download* de uma imagem de

disco no formato .iso e que ele gostaria de saber se os dados estão íntegros, que não tenha ocorrido nenhuma falha durante o processo, sem perder nenhum dado, ou que um software mal-intencionado instalado em seu computador não tenha modificado os dados, ou até mesmo ter realizado o *download* de um arquivo que é dito ser de tal origem, mas na realidade é de outra.

Em Bishop (2005) são apresentadas duas formas de garantia dessa propriedade. A primeira forma trata dos mecanismos de prevenção, que visam bloquear tentativas não autorizadas de mudar os dados e o uso de caminhos não autorizados para mudar os dados. A segunda forma são mecanismos de detecção que informam que a integridade dos dados não é mais confiável, podendo usar recursos do sistema para gerar alertas com informações da causa da corrupção dos dados.

Segundo a Shirey (2000, p. 17) disponibilidade é a “propriedade de um sistema ou um recurso do sistema ser acessível e utilizável mediante demanda por uma entidade do sistema autorizada, de acordo com as especificações de desempenho do sistema.” Quando há disponibilidade, usuários ou serviços autorizados devem conseguir acessar dados, serviços e sistemas (PFLEEGER; PFLEEGER, 2007). Por exemplo, se um usuário paga por um serviço de *streaming de dados*, como serviços de transmissão de vídeos, filmes e séries, e no contrato com o sistema que promove o *streaming* é descrito que este deve estar disponível em dia e hora específico, o serviço deveria estar. Existem inúmeros problemas que podem tornar esses recursos indisponíveis, como desastres naturais, problemas de conexão de servidor, erros de desenvolvimento dos sistemas e indisponibilidade por quantidade de acessos maior que a suportada.

Um dos ataques mais conhecidos atualmente é o *Distributed Denial of Service* (DDOS), ataque de negação de serviço, que tem por objetivo tornar um sistema indisponível por meio da realização de acessos e envio de pacotes em larga escala à sistemas, com o intuito de ultrapassar a capacidade de processamento dos servidores desses sistemas, tornando-os lentos e impossíveis de se utilizar durante o ataque, ou até mesmo derrubando o serviço.

2.2.2 Vulnerabilidade e Ameaça

Para planejar a segurança de aplicações web, estabelecer medidas de prevenção, proteção à ataques e em realizar testes de intrusão para verificar a segurança de um sistema, é necessário saber quais vulnerabilidades e ameaças estão presentes.

Conforme OWASP (2014, p. 27) “uma vulnerabilidade é uma falha ou fraqueza no projeto, implementação, operação ou gerenciamento de um sistema que poderia ser explorada para comprometer os objetivos de segurança do sistema”. Atualmente são desenvolvidos softwares cada vez mais complexos, com mais requisitos de negócio e com milhares de linhas de código, e combinando isso com o fato dos desenvolvedores não terem muito conhecimento em segurança ou não terem tempo hábil para planejá-la, acabam gerando cada vez mais problemas em seus sistemas, que podem tornar-se vulnerabilidades de segurança (UTO; MELO, 2009).

“Uma ameaça é qualquer coisa (um atacante externo malicioso, um usuário interno, uma instabilidade do sistema) que pode prejudicar os ativos de uma

aplicação (como os dados num banco de dados), explorando uma vulnerabilidade”, conforme OWASP (2014, p. 27).

Exemplificando num cenário, pode-se ter uma aplicação web que apresenta uma vulnerabilidade não corrigida e gerada devido ao desenvolvimento não ter adotado um padrão de projeto para tratamento dos dados vindos através da requisição do cliente ao servidor, que permite que sejam obtidos dados sigilosos do banco de dados desse sistema. Como ameaça temos um agente mal-intencionado que caso descubra essa vulnerabilidade, pode realizar um ataque e obter os dados confidenciais.

2.2.3 Ataques

Segundo a Shirey (2000, p. 12) um ataque é “uma investida à segurança de um sistema, derivado de uma ameaça inteligente, isto é, um ato inteligente que é uma tentativa deliberada (especialmente no sentido de um método ou técnica) de burlar os serviços de segurança e violar a política de segurança de um sistema”.

Um ataque viola a segurança de um sistema. Trata-se de uma tentativa, com sucesso ou não, de se obter informações ou tirar proveito de recursos, onde um agente malicioso, conhecido por atacante, tenta descobrir vulnerabilidades num sistema para depois utilizá-las.

Um atacante pode se utilizar de diversos meios para tentar obter qualquer tipo de informação que permita a ele ter acesso a um sistema protegido. Pode realizar ligações telefônicas para funcionários de uma empresa, para obter

informações, fingindo ser alguém importante na organização. Pode fazer uma varredura das portas de um servidor para encontrar brechas de acesso.

Pode-se classificar os ataques em dois grupos: os ataques passivos e ativos.

Ataques Passivos: São realizados de modo que não atrapalhe o funcionamento normal de um sistema, pois seu objetivo é simplesmente coletar e monitorar informações que são trocadas entre servidores, redes e aplicações. Stallings (2008) cita dois exemplos de ataques passivos: ataque de liberação de conteúdo da mensagem quando um atacante lê o conteúdo de uma mensagem enviada entre dois nodos numa rede, sem interferir no envio da mensagem; Ataque de análise de tráfego, no qual mesmo que o conteúdo seja transmitido de modo cifrado, é possível descobrir informações como origem e padrões.

Ataques Ativos: São ataques nos quais há adulteração em dados e modificação em requisições, podendo até deixar um serviço indisponível. Conforme Stallings (2008) esses ataques podem ser divididos em:

- Ataque de disfarce, no qual alguém pode conseguir validar sua identidade como se fosse outra pessoa e se autenticar num sistema, roubando ou alterando dados;
- Ataque de repetição, no qual ocorre a retransmissão de uma informação válida;
- Ataque de modificação de mensagens, neste ataque uma mensagem, enviada entre um usuário a outro, pode ser alterada de modo que o usuário

que a recebe tome alguma decisão incorreta, como alterar permissões de acesso;

- Ataque de negação de serviço, geralmente realizado através de inúmeras requisições à servidores, vindos de diferentes nodos conectados à Internet, com o intuito de torná-los indisponíveis.

2.3 OPEN WEB APPLICATION SECURITY PROJECT

A OWASP (*Open Web Application Security Project*) é uma comunidade mundial livre e aberta cujo objetivo é promover o conhecimento a respeito de segurança de software. No documento da OWASP (2014) é comentado que sua missão é tornar visível informações a respeito da segurança de aplicações, para que pessoas e organizações possam tomar decisões e analisar os riscos de segurança que seus sistemas possam enfrentar. Por se tratar de uma comunidade aberta, todos são livres para participar dos projetos que estão disponíveis no próprio site da comunidade, que segue o formato de wiki.

O trabalho da comunidade recebe o suporte da fundação OWASP, uma organização sem fins lucrativos estabelecida nos Estados Unidos em 2004. Como não é filiada a nenhuma organização ou empresa de tecnologia e não enfrenta pressões comerciais, consegue promover a divulgação de informações imparciais sobre a segurança de aplicações web.

Para o desenvolvimento deste trabalho é utilizada como base a metodologia de testes de intrusão de um dos projetos da OWASP, o *OWASP Testing Guide v4*. Nesta seção são explicadas algumas das técnicas de teste de segurança propostas

pela metodologia OWASP, enfatizando testes de intrusão que é a técnica adotada neste trabalho e explicando quais tipos de teste foram selecionados para compor o guia desenvolvido.

2.3.1 Técnicas de Teste

2.3.1.1 Revisão e Inspeção de Manuais

Segundo a Owasp (2014), inspeção de manuais são revisões realizadas em cima de documentos que abordam a segurança de pessoas, políticas e processos adotados por uma organização em seu ciclo de desenvolvimento de software, que devem fazer parte de um programa de testes adotados.

Os manuais a serem analisados são documentos que contêm informação a respeito da arquitetura de uma aplicação, incluindo quais tecnologias foram adotadas para seu desenvolvimento, regras de controle de acesso ao sistema e visualização de informações, requisitos funcionais, não funcionais e de segurança e padrões de desenvolvimento adotados para implementação de código seguro.

2.3.1.2 Modelagem de Ameaças

A modelagem de ameaças permite aos arquitetos de aplicações identificarem possíveis ameaças antes mesmo do software estar desenvolvido, pois fornece a visão de quais comportamentos um atacante poderia ter para tentar de alguma forma realizar seu objetivo quanto ao sistema, seja tornar a aplicação indisponível aos seus usuários ou ter acesso não autorizado a informações sensíveis.

Conforme a Owasp (2014), modelagem de ameaças pode ser considerada também uma avaliação de riscos e permite a quem desenvolve e estrutura a aplicação, mitigar potenciais vulnerabilidades, desenvolver soluções que permitam a recuperação do sistema após a ocorrência de problemas e focar recursos nas áreas mais críticas do sistema. Braga (2007, p. 71) diz que “o modelo de ameaças é uma ferramenta de projeto que expressa como a arquitetura do sistema funciona diante de ataques”.

2.3.1.3 Revisão do Código Fonte

Processo no qual o profissional de segurança ou desenvolvedor com habilidades em segurança faz uma análise do código fonte de uma aplicação web em busca de possíveis problemas que possam comprometer um sistema. Segundo a Owasp (2014, p. 13), “muitas vulnerabilidades de segurança sérias não podem ser encontradas com qualquer outra forma de análise ou teste”.

Problemas de controle de acesso, erros na modelagem da lógica do negócio e a utilização de algoritmos criptográficos de forma inadequada servem de exemplo de problemas que podem ser detectados na revisão de código (OWASP, 2014).

2.3.1.4 Testes de Intrusão

Técnica onde testers de segurança buscam com o auxílio de ferramentas de automatização, vulnerabilidades de segurança numa aplicação que esteja em execução, sem precisar conhecer seu código fonte.

Segundo a Owasp (2014, p. 14), “os testers atuam como atacantes e tentam encontrar e explorar vulnerabilidades”. Weidman (2014, p. 30) afirma que “Testes de invasão ou *pentesting* envolvem a simulação de ataques reais para avaliar os riscos associados a potenciais brechas de segurança”, também comenta que os testers exploram as vulnerabilidades encontradas para verificar quais recursos poderiam ser obtidos pelos atacantes ao explorar esses problemas.

Nenhum sistema pode ser considerado 100% seguro e sabendo disto, empresas e organizações devem planejar meios de buscar e explorar vulnerabilidades que possam existir em seus sistemas, a fim de encontrar soluções. Durante o processo de *pentest*, o profissional que o estiver realizando deve elaborar um relatório contendo as vulnerabilidades encontradas, para que os desenvolvedores possam adotar as correções necessárias na aplicação web.

Conforme a Owasp (2014), existem vantagens ao realizar um teste de invasão, como o tempo menor para executá-lo, graças ao auxílio das ferramentas automatizadas de teste, nível de conhecimento menor em segurança pelo profissional que está executando o teste. Como desvantagem, os testes só podem ser realizados quando o estado do software já está num nível avançado do ciclo de desenvolvimento, já que é necessário que a aplicação esteja rodando para que o *pentest* possa ser executado.

Pode-se dividir o teste de invasão em três tipos:

- Teste de caixa preta;
- Teste de caixa cinza;
- Teste de caixa branca.

O teste de caixa preta é o processo no qual a equipe de testes não tem conhecimento profundo a respeito da aplicação, sem conhecer manuais, código fonte ou informações técnicas. Peixinho e Fonseca (2013) comentam “esse tipo de teste foi projetado para oferecer o teste de penetração mais realístico possível.”

No teste de caixa cinza o tester toma conhecimento de algumas informações e detalhes técnicos da aplicação que será submetida ao teste. Geralmente é realizado quando o intuito de uma organização é descobrir se em posse de tais informações, um atacante poderia encontrar e explorar alguma vulnerabilidade do sistema.

No teste de caixa branca o tester tem a maior quantia de informações antes de realizar o teste. Normalmente é usado para simular o ataque realizado por fontes internas a uma organização. Informações sobre o servidor da aplicação web, *schemas* do banco de dados, endereços de ip, código fonte, usuários do sistema e até documentos ou manuais técnicos da aplicação, podem ser de conhecimento do atacante, portanto também serão usados nesse processo de teste.

2.3.2 Tipos de Teste

Nesta seção são definidos os dois tipos de teste da metodologia OWASP que são cobertos por este guia, com o intuito de auxiliar numa maior compreensão dos testes que podem ser realizados por quem utilizar este trabalho.

2.3.2.1 Teste de Autenticação

Autenticação é um processo que tem por objetivo provar que algo é verdadeiro ou confirmar a identidade de alguém. “Na segurança computacional, autenticação é o processo de tentar verificar a identidade digital de alguém que deseja se comunicar” (OWASP, 2014, p. 66).

O ato de um usuário tentar efetuar *login* em um sistema por meio de nome do usuário e senha é um processo de autenticação, no qual a aplicação em questão verificará se os dados informados são válidos e fornecerá ou não acesso aos seus recursos.

Mecanismos como assinatura e certificado digital, permitem que mensagens e documentos, enviados seja por e-mail ou por mecanismo de *upload* em aplicações web, possam ter sua autenticidade verificada. Os mecanismos também permitem o estabelecimento de comunicação entre o cliente de uma aplicação e seu servidor, por meio de protocolos criptográficos seguros.

Testes de autenticação focam seus esforços basicamente em verificar o funcionamento desses mecanismos, tentando obter algum tipo de informação, com o intuito de que seja possível efetuar uma autenticação falsa (OWASP, 2014).

2.3.2.2 Teste de Validação de Entrada

Segundo OWASP (2014), a maioria das vulnerabilidades principais de aplicações web são provenientes da validação inadequada de entrada dos dados vindos do cliente ou serviço que se comunica com a aplicação. Vulnerabilidades

como *SQL injection*, *cross-site scripting* e *buffer overflows* são exemplos de problemas de segurança que acontecem devido a forma como os dados de uma aplicação estão sendo tratados.

Aplicações web muito grandes apresentam vários formulários com campos para entrada de dados, figura 2. Pode-se citar como exemplo uma aplicação voltada para a área da saúde e nela teríamos a possibilidade de cadastrar dados de profissionais ou pacientes, e somente nesses dois cadastros já existiriam inúmeros campos que precisam de uma série de validações, para então poder enviar os dados ao servidor para serem salvos numa base de dados.

O teste de validação de entrada é realizado por meio de um conjunto de ações que visam verificar se as validações realizadas, em cima das entradas de dados da aplicação web, são suficientes, conforme OWASP (2014, p. 98).

Dados de identificação

CPF *

Nome completo *

Data de nascimento *

Sexo *

☐ Masculino ☐ Feminino

Endereço

CEP *

UF *

Cidade *

Bairro *

Logradouro *

Número *

☐ Sem número

Complemento *

Figura 02 - Modelo de formulário de cadastro

2.4 SEGURANÇA DE APLICAÇÕES WEB

Com a popularização da web nos últimos anos e o aumento de pessoas conectadas à rede, seja por meio de computadores ou dispositivos móveis, muitos empreendedores viram grandes oportunidades de negócio que poderiam ser realizadas nesse ambiente. As empresas têm focado cada vez mais na criação de aplicações web que satisfaçam as necessidades dos usuários.

Existem sistemas complexos de e-commerce, através dos quais inúmeros clientes podem comprar os mais variados produtos, tudo através da Internet, diretamente de seu celular. Gerenciadores de banco que possibilitam a realização de transações financeiras como transferência eletrônica. Prontuário eletrônico no qual médicos cadastram todas as informações clínicas de um atendimento a seu paciente. Aplicativos de smartphones que permitem a publicação de fotos, texto e vídeos, bem como a comunicação entre diversas pessoas em qualquer lugar do mundo.

Essas tecnologias trazem inúmeros benefícios para nosso cotidiano, porém concentram informações sensíveis. Stuttard e Pinto (2011) comentam que se for encontrada uma vulnerabilidade crítica num desses sistemas é possível conseguir acesso a dados sensíveis irrestritamente. Um atacante poderia ter acesso a números de cartão de crédito, a senhas pessoais ou documentos sigilosos.

Devido a isso a atenção quanto a segurança das aplicações web deve ser uma das principais preocupações. Segundo OWASP (2014, p. 10) “um dos melhores métodos para evitar que erros de segurança aconteçam em aplicações de produção é melhorar o ciclo de vida de desenvolvimento de software, incluindo a

segurança em cada uma de suas fases”, desde a etapa de definição até a implantação.

Sabe-se que as aplicações web geralmente possuem muita interação com o usuário, através de preenchimento de formulários, realização de eventos e requisições ao servidor, e geralmente nessa conversa entre cliente e servidor pode ocorrer a troca de muitas informações, e conforme afirmam Stuttard e Pinto (2011) esses sistemas devem sempre considerar que é possível usar de forma maliciosa qualquer entrada de dados no servidor e que a maioria dos ataques acontecem explorando essas entradas. Deve-se realizar validações, controle de acesso e outros mecanismos de proteção, com o intuito de evitar que um atacante submeta dados que lhe permitam explorar alguma vulnerabilidade.

Neste capítulo serão descritas as principais vulnerabilidades presentes no documento da OWASP Top 10 que serão objeto de estudo nos cenários controlados presentes no capítulo de desenvolvimento. As vulnerabilidades foram escolhidas devido a sua prevalência nas aplicações web (OWASP, 2013).

2.4.1 Injeção de Código

OWASP (2013) afirma que as falhas de Injeção, tais como injeção de comandos SQL (*Structured Query Language*), ocorrem quando um atacante envia dados manipulados através de comandos ou consultas a um interpretador responsável por executá-los, tentando executar comandos ou obter controle de acesso quando ocorre o processamento desses dados.

Pode-se por exemplo, citar os casos de uso da técnica de *Sql Injection* e que sua principal causa é a concatenação de dados fornecidos por um usuário numa string que armazena um comando SQL a ser executado por um sistema gerenciador de banco de dados (OWASP, 2014). Para evitar essa injeção bastaria tratar essas strings, evitando o uso de caracteres especiais, ou adotando tecnologias mais recentes e seguras para construção desses comandos.

Um exemplo clássico de ataque com a técnica de injeção SQL é um formulário de autenticação que não apresenta filtro de caracteres especiais ou não faz uma a validação dos dados de modo correto, conforme a figura 3.

A imagem mostra um formulário de autenticação simples. No topo, o título "Login" está em negrito. Abaixo dele é um campo de entrada retangular com o texto "Login" em cinza. Logo abaixo, o título "Senha" está em negrito, seguido por outro campo de entrada retangular com o texto "Senha" em cinza. Na base do formulário, há um botão retangular azul com o texto "Entrar" em branco.

Figura 03 - Formulário padrão de autenticação

Ao clicar no botão de entrar, uma requisição é disparada ao servidor da aplicação, passando como entrada de dados o *login* e a senha. A aplicação utiliza os dados diretamente, sem realizar nenhum tratamento, no comando SQL presente na figura 4, que será executado para buscar o usuário no banco de dados. Se for

encontrado algum usuário com o mesmo *login* e senha, o usuário estará autenticado na aplicação.

```
SELECT * FROM USUARIO WHERE LOGIN='$login' AND SENHA='$senha'
```

Figura 04 - Consulta SQL de usuário

Como os dados não passam por uma validação adequada, o comando sql deverá ficar como na figura 5, se for informado como valor de *login* a *string* de ataque ' or 1=1; --.

```
SELECT * FROM USUARIO WHERE LOGIN='' or 1=1; --' AND SENHA=''
```

Figura 05 - Consulta SQL com *string* de ataque

A *string* de ataque é o valor digitado no campo de *login*, criada com o intuito de alterar o comando SQL executado pela aplicação. Nesse exemplo, a expressão or 1=1 fará que a condição do comando SQL seja sempre verdadeira, retornando todos os usuários. Para comentar o restante do código sql e impedir sua execução, os símbolos -- são usados e o parâmetro senha será desconsiderado no comando.

Ao retornar todos os usuários com o comando SQL, a aplicação realizará a autenticação de um usuário, com isso o atacante conseguiu enganar o sistema de autenticação da aplicação.

Como detectar: Analisar se o ambiente que irá executar comandos ou receber dados para a execução dos comandos está tratando adequadamente as

entradas. Segundo a Owasp (2013) os dados não-confiáveis devem ser separados desses comandos. Pode-se também realizar a revisão de código com o intuito de identificar problemas na construção do comando SQL, como a concatenação direta de dados fornecidos pelo usuário, e usar ferramentas de análise de código que verificarão se existe alguma brecha para inserir comandos ou dados em alguma funcionalidade da aplicação (OWASP, 2013).

Uma vez que forem detectados problemas, é possível realizar testes de intrusão para validar e mapear a sequência de passos para a exploração da vulnerabilidade e a partir disso tomar medidas corretivas. Uto e Melo (2009, p. 258) afirmam que “a melhor maneira de realizar testes em uma aplicação, para detectar se são vulneráveis à injeção de SQL, é por meio de ferramentas automatizadas.”

Como evitar: Segundo Uto e Melo (2009) pode-se realizar um conjunto de ações que podem evitar esse tipo de ataque e entre elas estão:

1. Realizar validações quanto aos valores fornecidos pelos usuários, como tamanho do campo, uso de caracteres especiais, entre outras regras.
2. Além de ser uma forma muito legada de se escrever consultas SQL, concatenar entradas de dados em *strings* é extremamente perigoso, como mostrado na figura 4, logo não devemos adotar essa prática.
3. Deve-se usar comandos que são pré-compilados e que definem as posições dos parâmetros a serem executados, evitando que um parâmetro altere uma consulta de modo a ter outro comportamento.
4. Deve-se sempre realizar o tratamento de exceções, evitando que erros que contenham comandos SQL sejam exibidos ao usuário.

5. Estabelecer privilégios mínimos às contas de usuários que sejam utilizadas na aplicação, evitando a permissão de execução de comandos DDL (*Data Definition Language*), para que em caso de sucesso um atacante não possa modificar a estrutura do banco de dados.

2.4.2 Quebra de autenticação e Gerenciamento de Sessão

As aplicações web apresentam muitos recursos disponíveis e alguns só podem ser acessados por categorias específicas de usuários, também em muitos casos, apresentam a necessidade de autenticação com o sistema para que suas funcionalidades sejam habilitadas.

Pode-se ter um sistema onde os usuários apresentam perfis e cada um deles tem um conjunto de permissões para a execução e acesso a módulos ou simplesmente estabelecer uma conexão entre cliente e servidor com determinado tempo de validade e ainda de maneira mais básica dar acesso a somente usuários cadastrados por meio de *login* e senha. Esses são exemplos básicos dos mecanismos de autenticação que podem ser utilizados.

Como requisitos de segurança no acesso às aplicações são utilizados em conjunto os mecanismos de autenticação, gerenciamento de sessão e controle de acesso (STUTTARD; PINTO, 2011).

O resultado de uma autenticação realizada com sucesso é um conjunto de credenciais que estabelecem a sessão de um usuário. Essas credenciais podem ser *tokens* ou *cookies*, que são valores gerados pelo servidor da aplicação utilizando algoritmos criptográficos (OWASP, 2014). Nas figuras 6 e 7 é possível ver o

processo de *login* no sistema da Google e as credenciais geradas após autenticação.

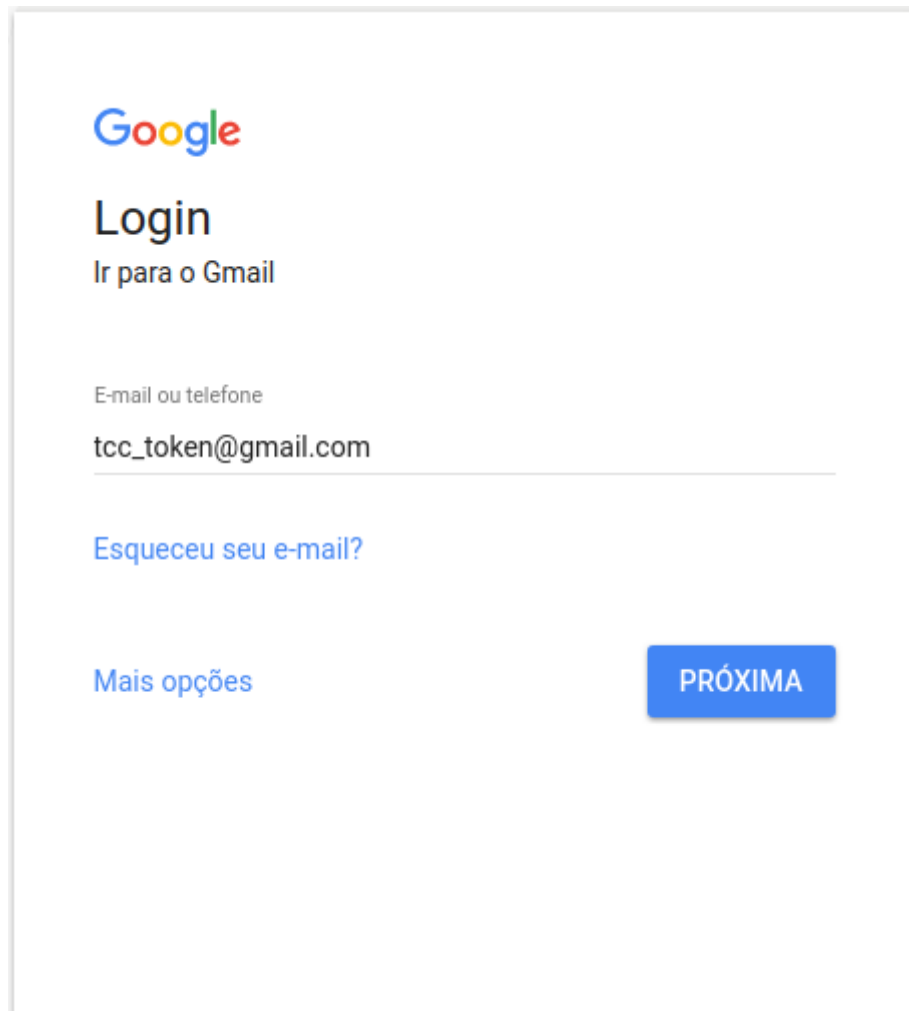


Figura 06 - Tela de autenticação do Google

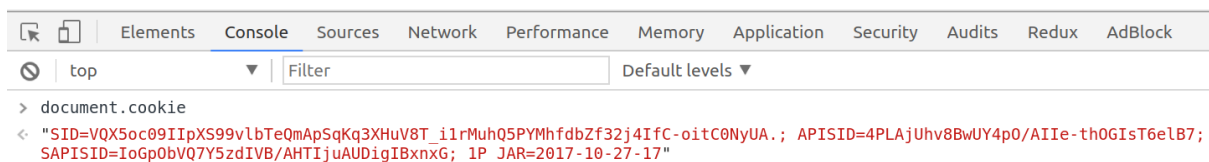


Figura 07 - Credenciais de autenticação

No documento Top 10 da Owasp (2013) pode-se ver que a autenticação e gerenciamento de sessão costumam ser implementados de forma personalizada e muitas vezes incorretamente, utilizando *tokens* fixos ou com baixa aleatoriedade, *tokens* sem data de expiração ou parâmetros booleanos de autenticação, gerando falhas na gestão das senhas do usuário, no tempo de expiração de sessão e tokens, dando oportunidade aos atacantes de assumir a identidade de outros usuários.

Como detectar: Os recursos de autenticação e gerenciamento de sessão podem estar vulneráveis se (OWASP TOP 10, 2013):

1. Os dados de autenticação de um usuário que são salvos não são protegidos por hash ou criptografia, ou protegidos por algoritmos vulneráveis;
2. Funcionalidades que envolvem o gerenciamento das contas de usuários permitem a descoberta de suas credenciais;
3. IDs de sessão são expostos na URL;
4. É possível realizar ataques de fixação de sessão aos IDs de sessão
5. Tokens, IDs, e a própria sessão não expiram, mesmo após o tempo de validade;
6. IDs de sessão não são renovados após o login bem-sucedido;
7. Credenciais e informações privadas de sessão são transmitidas por conexões não criptografadas.

Como evitar: Conforme Uto e Melo (2009) diversas medidas podem ser adotadas para obter mecanismos de autenticação e gerenciamento de sessão mais robustos, são elas:

1. Desabilitar contas padrão de todos os recursos utilizados pelo sistema, como a plataforma de execução, banco de dados, servidor web e sistemas operacionais;
2. Implementar políticas de senhas fortes, especificando tamanho mínimo, complexidade, tempo de validade, bloqueio por quantidade de tentativas inválidas e histórico de tentativa de acessos;
3. Transmitir credenciais e identificadores de sessão somente por canais protegidos criptograficamente;
4. Não confiar em informações disponíveis aos usuários, para verificar se eles estão autenticados ou não;
5. Sempre verificar se um usuário está autenticado e se possui privilégios antes de atender qualquer requisição a recurso protegido;
6. Estabelecer mecanismos de perguntas secretas com respostas mais complexas para módulo de recuperação de senha, limitar o número de tentativas evitando ataque de força bruta e em caso de sucesso, enviar link por e-mail para redefinição da senha;
7. Sempre solicitar a senha atual para realizar a redefinição de senha;
8. Usar tecnologias de gerenciamento de sessão de bibliotecas mais recentes e testadas, evitando usar mecanismos próprios;
9. Verificar se identificadores de sessão tem boa aleatoriedade e não aceitar se forem definidos pelo próprio usuário, gerando em resposta um novo identificador;
10. Deve-se encerrar as sessões em casos que o usuário se torne ocioso;

11. O token ou identificador de sessão deve ser invalidado após o tempo de validade ou quando o usuário fizer logoff da aplicação.

2.4.3 Cross-Site Scripting

Trata-se de uma vulnerabilidade também conhecida por XSS, que permite a um atacante injetar código ou informações e executar scripts no navegador do cliente da aplicação. Geralmente ocorre devido a falta de validações ou filtros dos dados que o servidor recebe e que são usados para gerar páginas dinâmicas. Como o código injetado nessas páginas dinâmicas são válidos e interpretados por um navegador, o cliente que acessar a página sofrerá com a execução do script que foi injetado, podendo ter sua sessão e dados comprometidos (UTO; MELO, 2009).

Existem alguns tipos de vulnerabilidades de *Cross-Site Scripting*, as que serão abordadas neste trabalho são XSS Persistido e XSS Refletido:

XSS Persistido: Segundo a OWASP (2017) esse é considerado o tipo mais perigoso, visto que os dados inseridos pelo atacante na aplicação geralmente são armazenados no banco de dados e muitos usuários podem recuperar esses dados ao usar a aplicação, obtendo na realidade um código malicioso que será executado em seu navegador.

Para exemplificar, pode-se considerar uma página acessada por administradores de um sistema e que fornece os dados de um usuário pesquisado, conforme figura 8. Quando um usuário é cadastrado nesse sistema o campo e-mail

não recebe nenhum tratamento, não apresenta validações e nem filtragem de caracteres especiais.



Informações de usuário

Nome

Login

E-mail

Figura 08 - Formulário para visualização dos dados de usuário

O código HTML da página referente ao campo e-mail está na figura 9.

```
<input disabled type="text" name="email" value="admin@adm.com" />
```

Figura 09 - Campo HTML de e-mail

Se durante o cadastro do usuário Admin Tester o valor salvo no campo e-mail fosse a string de ataque `a@a.com"><script>alert('Teste de XSS Persistido')</script><!--`, o código HTML do campo e-mail deveria ficar conforme a figura 10, ao carregar os dados do usuário na tela.

```
<input disabled name="email" type="text" value="a@a.com"><script>alert("Teste de XSS Persistido")</script><!--" />
```

Figura 10 - Campo HTML com código injetado

Dessa forma, ao carregador os dados do usuário pesquisado na tela, o comando injetado será executado no navegador do usuário que realizou a busca e mostrará um alerta, figura 11.

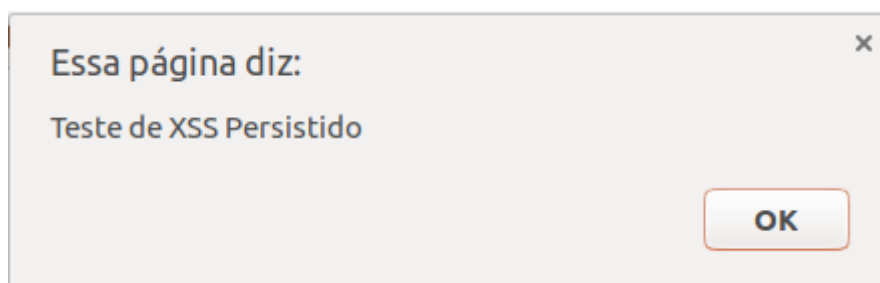
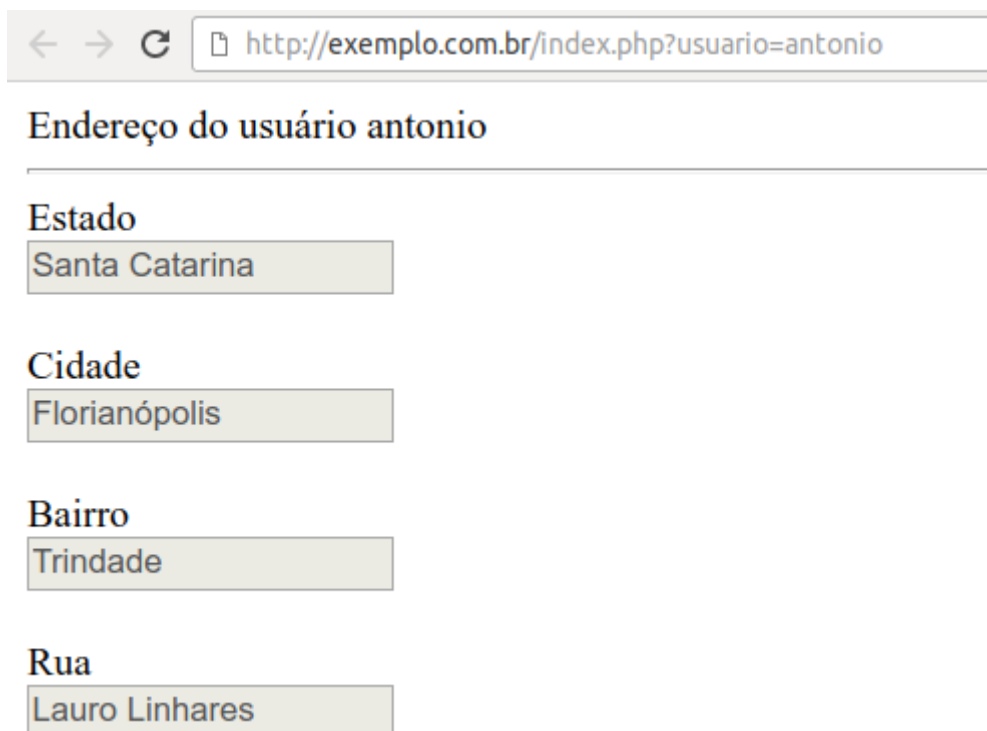


Figura 11 - Alerta gerado por injeção de código

XSS Refletido: Esse tipo de XSS ocorre quando a entrada do usuário é retornada pela aplicação como resposta imediata após a realização de uma requisição ao servidor pelo usuário, na qual os dados faziam parte da própria requisição, conforme OWASP (2017).

Pode-se entender melhor se imaginar uma aplicação simples que retorna o endereço de um usuário cadastrado. Ao acessar a URL da figura 12, passando como valor do parâmetro *usuario* o *login* do usuário cadastrado, o endereço deverá ser retornado. O problema é que a aplicação utiliza o valor passado no parâmetro *usuario* da requisição, no corpo da página.



A screenshot of a web browser window. The address bar shows the URL `http://exemplo.com.br/index.php?usuario=antonio`. Below the address bar, the text "Endereço do usuário antonio" is displayed. The form contains four input fields, each with a label above it: "Estado" with the value "Santa Catarina", "Cidade" with the value "Florianópolis", "Bairro" with the value "Trindade", and "Rua" with the value "Lauro Linhares".

Figura 12 - Busca de endereço do usuário

Se a entrada de dados usuario não receber nenhuma validação, ao executar a requisição da figura 13, um alerta deverá ser exibido no navegador, figura 14.

[http://exemplo.com.br/index.php?usuario=<script>alert\('Teste de XSS Refletido'\)</script>](http://exemplo.com.br/index.php?usuario=<script>alert('Teste de XSS Refletido')</script>)

Figura 13 - Url para ataque de XSS Refletido

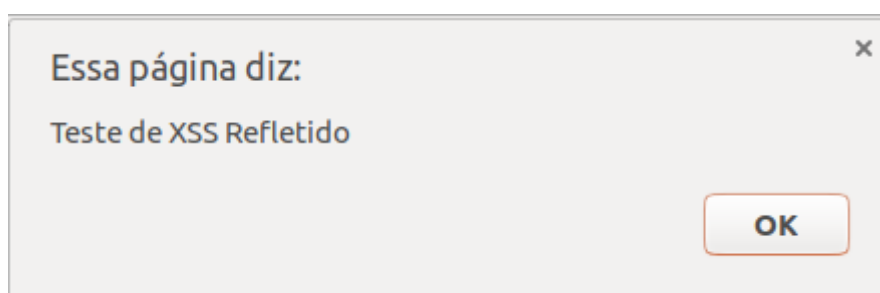


Figura 14 - Alerta de XSS Refletido

Como detectar: Verificar se todas as entradas de dados da aplicação estão sendo validadas corretamente, para tentar impedir que conteúdo contendo comandos e scripts possam ser enviados ao servidor e acabem sendo inseridos dinamicamente nas páginas que os usuários acessarem. Apesar de existirem ferramentas automatizadas que detectam problemas de XSS, cada aplicação pode usar tecnologias diferentes e modos diferentes de implementação, o que torna o trabalho dessas ferramentas bem mais difícil, portanto no processo de detecção é necessário a revisão manual de código, conforme a OWASP Top 10 (2013).

Como evitar: Uto e Melo (2009) dizem que para evitar essas vulnerabilidades sempre ao desenvolver uma aplicação é necessário considerar que toda informação vinda dos usuários é maliciosa e portanto todo campo ou parâmetro que permita que um dado seja enviado ao servidor deve passar por uma validação, analisando se a informação digitada representa um valor válido, se aceita somente os caracteres permitidos àquele campo e se seu tamanho é limitado, que são validações necessárias conforme as regras do domínio da aplicação.

3 FERRAMENTAS

Neste capítulo será fornecida uma breve descrição das ferramentas que são utilizadas nos testes realizados neste guia. A configuração de cada uma, será explicada no capítulo de Guia para Testes de Intrusão, no qual efetivamente serão utilizadas.

3.1 OWASP BROKEN WEB APPLICATIONS PROJECT

Conforme Willis (2017), trata-se de um projeto desenvolvido pela comunidade da OWASP, cujo objetivo é fornecer uma máquina virtual com uma coleção de aplicações web *open source* e vulneráveis, para que interessados na área de segurança possam realizar testes de intrusão seguros, permitindo na prática aprenderem mais e confirmarem a teoria.

Utilizando a máquina virtual OWASP Broken é possível aprender mais sobre segurança em aplicações web, realizar técnicas de testes manuais, utilizar ferramentas automatizadas de detecção de vulnerabilidades, testar ferramentas de análise de código e efetuar ataques controlados. Para a realização dos testes presentes neste guia, serão utilizadas aplicações vulneráveis que estão nesse projeto.

Essa máquina virtual divide seu conjunto de ferramentas em:

Aplicações de treinamento: Foram desenvolvidas com vulnerabilidades intencionais e que dão a oportunidade para quem deseja aprender mais a respeito de detectar e explorar problemas de segurança. Entre elas pode-se citar a famosa OWASP WebGoat e OWASP Mutillidae II (figura 15).

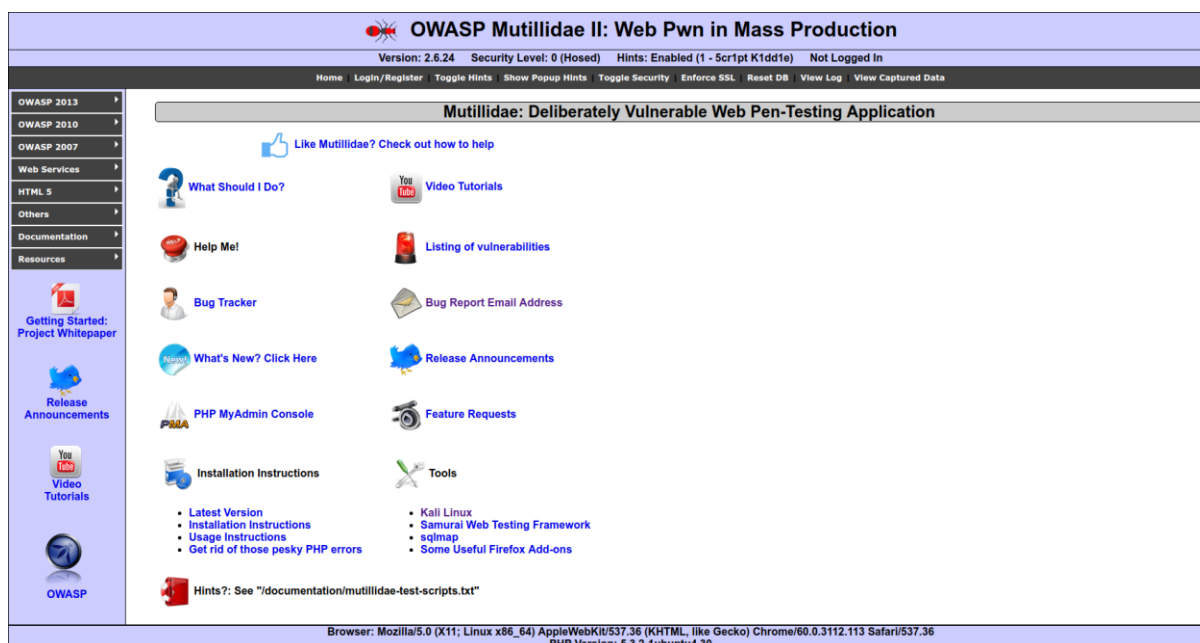


Figura 15 - Mutillidae

Aplicações realistas, intencionalmente vulneráveis: Aplicações que foram desenvolvidas para terem comportamento semelhante a aplicações reais, porém com uma série de vulnerabilidades intencionais. Como OWASP Vicnum e WackoPicko.

Versões antigas de aplicações reais: Entre as aplicações presentes na OWASP Broken, estas são as mais interessantes por serem versões antigas de aplicações que podem ainda estar no mercado e que anteriormente tiveram algumas vulnerabilidades. Como exemplos pode-se citar o WordPress, figura 16, em sua versão 2.0.0, GetBoo e Joomla.

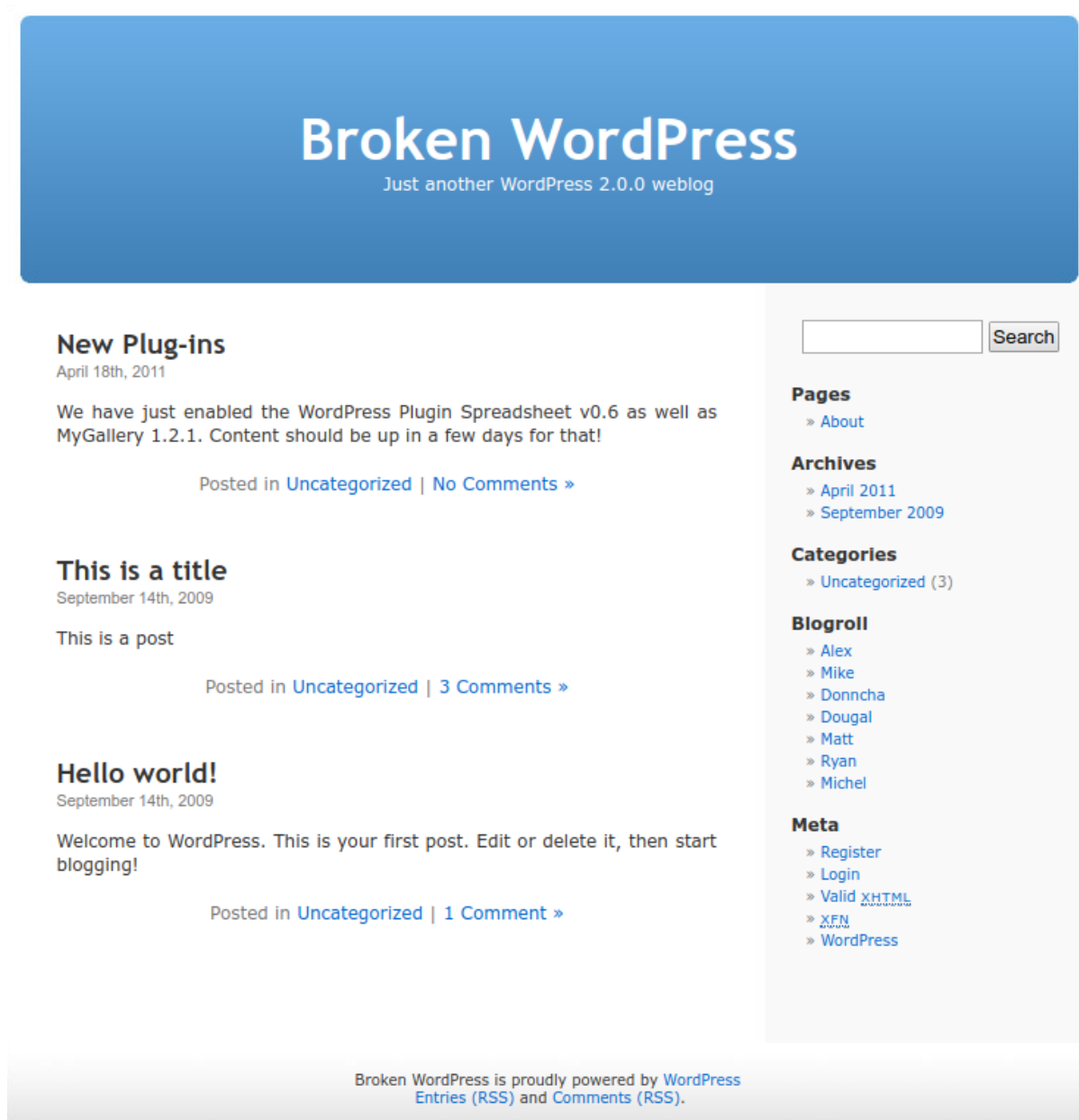


Figura 16 - WordPress

Aplicações para testes de ferramentas: Aplicações que foram desenvolvidas de modo a permitir a detecção de vulnerabilidades por meio de analisadores web, como *scanners* de segurança de aplicações web. Uma delas é a OWASP Zap-Wave.

Pequenas aplicações e páginas para demonstração: Aplicações para demonstração de conceitos específicos, como Mandiant Struts Forms.

3.2 KALI LINUX

Segundo a *Offensive Security* (2017), trata-se de uma distribuição Linux baseada em Debian, que apresenta centenas de ferramentas especificamente para a realização de testes de penetração e auditoria de segurança. Através delas é possível estudar e realizar forense computacional, testes de intrusão e aprender a respeito de segurança. Durante a execução dos testes do guia serão utilizadas ferramentas presentes na Kali Linux para a detecção de vulnerabilidades em aplicações.

Esse sistema operacional, figura 17, apresenta ferramentas para análise de vulnerabilidades, *scan* de portas de servidores, detecção de vulnerabilidades em aplicações web, ataques às senhas como força bruta, ferramentas de *sniffer* para monitorar e analisar o tráfego de rede, injeção de código malicioso, entre outras, contando com mais de 600 ferramentas de teste de intrusão.

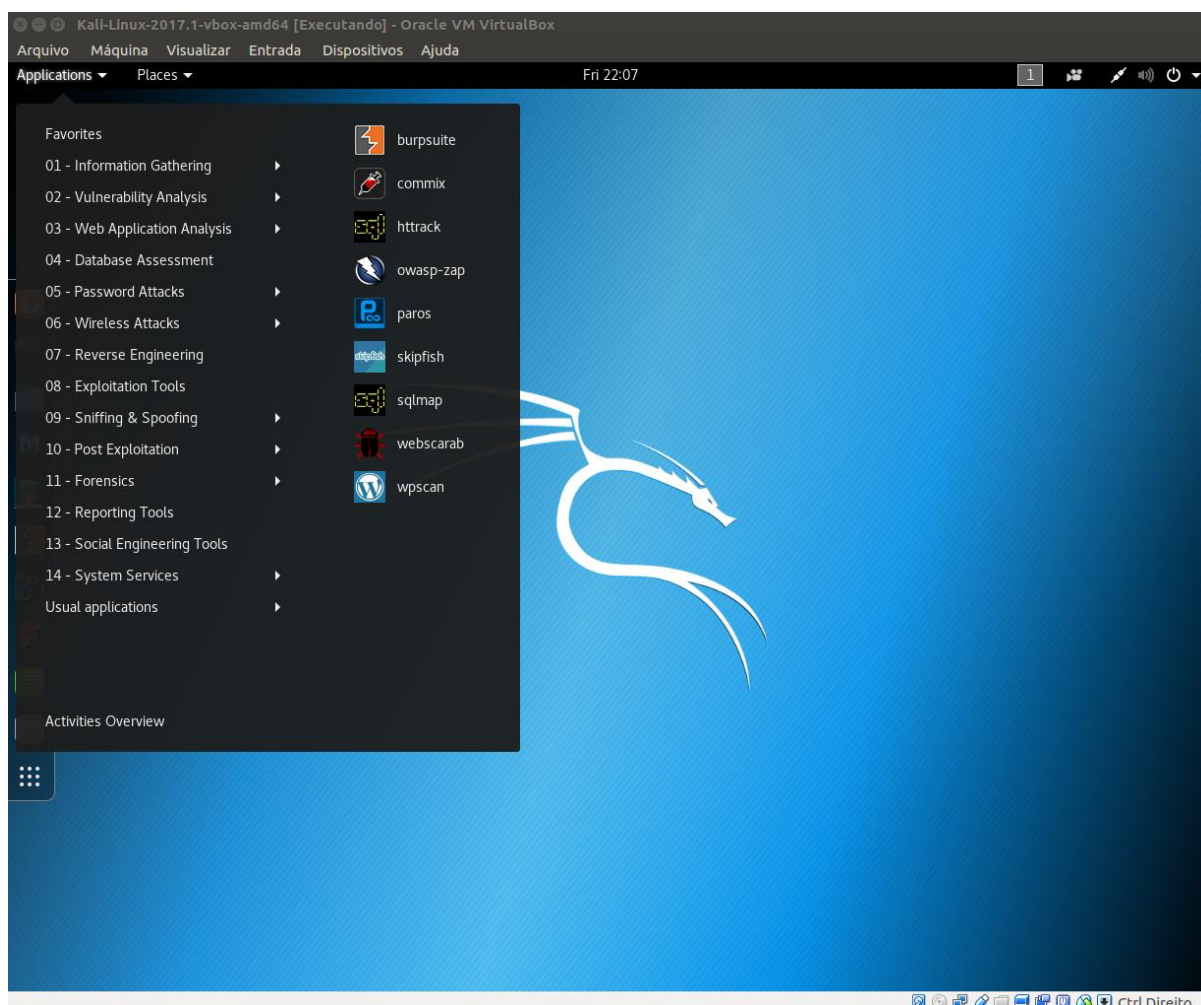


Figura 17 - Kali Linux no virtualbox

3.3 OWASP ZED ATTACK PROXY

Mais conhecida como OWASP Zap, ver figura 18, é uma ferramenta de segurança automatizada mantida por centenas de voluntários e sua principal funcionalidade é realizar *scan* em aplicações web em busca de vulnerabilidades. Bennetts (2017) comenta que pode ser usada tanto por especialistas em segurança como desenvolvedores ou testers que estejam começando na área de testes de intrusão. Ela é uma ferramenta *open source*, fácil de usar e instalar, multi plataforma e traduzida em diversos idiomas.

Devido a sua facilidade de uso, essa ferramenta é adotada nos testes realizados neste guia, permitindo a identificação de vulnerabilidades nas aplicações e a partir disso a realização dos ataques. Ela pode ser usada tanto numa aplicação em produção como em fase de testes.

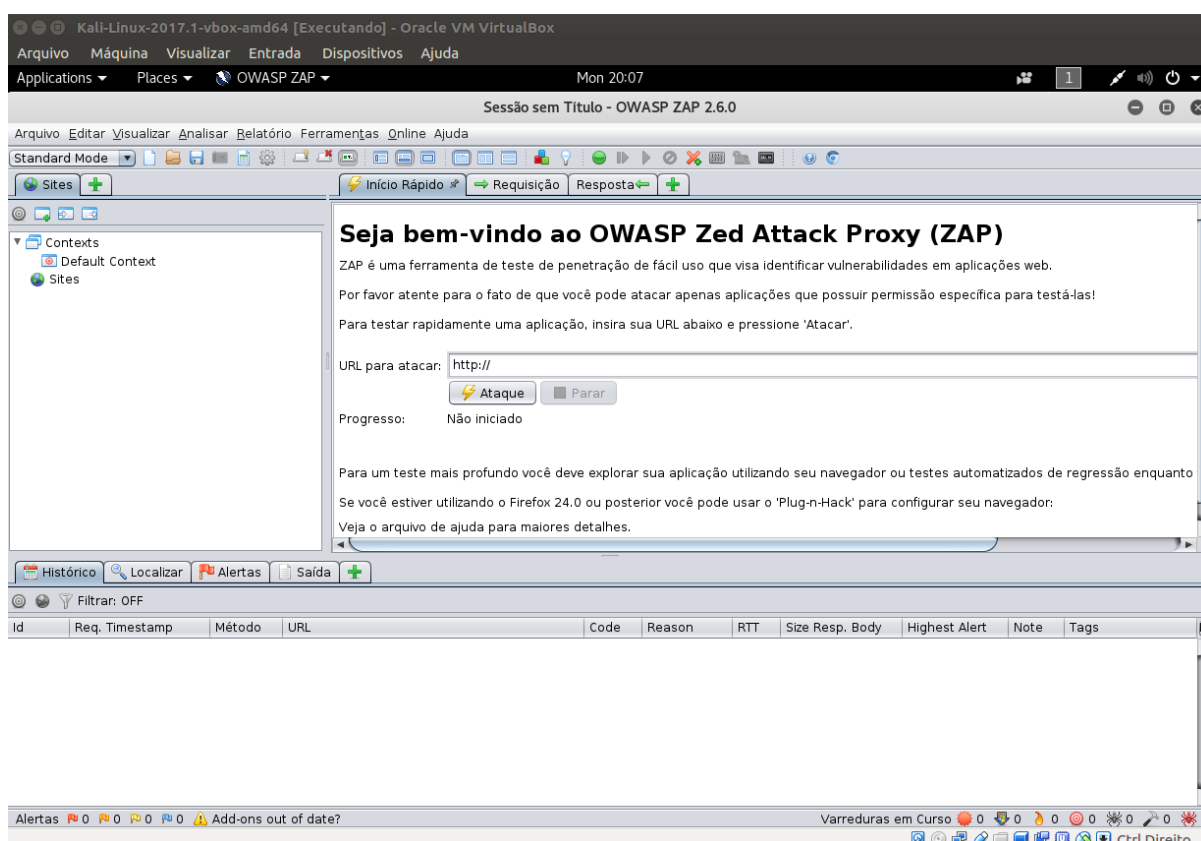


Figura 18 - OWASP Zap

4 GUIA PARA TESTES DE INTRUSÃO

Neste capítulo será ensinada detalhadamente a parte prática do guia de segurança proposto por este trabalho. Inicialmente será mostrado como configurar todo o ambiente necessário para executar os testes presentes neste guia e

derivados da metodologia do *Owasp Testing Guide*. Após a configuração são realizadas varreduras por vulnerabilidades em aplicações web selecionadas de exemplo, para então realizar os testes de segurança com o intuito de explorar os problemas encontrados.

4.1 PROPOSTA

Para alcançar os objetivos deste trabalho serão executados testes de intrusão, legalmente, em aplicações que apresentam vulnerabilidades e disponíveis para estudos de segurança em sistemas web.

As aplicações foram escolhidas de modo a permitir a descoberta e exploração de vulnerabilidades de injeção de código, autenticação e cross-site scripting, descritas no documento *OWASP Top-Ten*. Essas vulnerabilidades foram selecionadas por serem as mais prevalentes nos dados da OWASP e por apresentarem maior risco em aplicações web.

Os testes de intrusão do tipo de validação de entrada e autenticação, da metodologia do *Owasp Testing Guide*, foram escolhidos pois conseguem detectar e explorar as vulnerabilidades selecionadas.

Em cada teste é ensinado e demonstrado como encontrar uma vulnerabilidade numa aplicação web, utilizando uma ferramenta de *scan* conhecida por Owasp Zap. Após encontrar a vulnerabilidade será ensinado o passo-a-passo para concretizar o ataque.

A ferramenta Owasp Zap será executada através da máquina virtual Kali Linux, que estará numa VPN conectada com outra máquina virtual com a Owasp

Broken. O processo de descoberta de vulnerabilidades será a partir do uso da Owasp Zap, que realizará o *scan* das aplicações web que estarão em execução na Owasp Broken.

Os testes de intrusão podem ser executados no navegador de uma máquina que tenha acesso às aplicações presentes na máquina virtual. Através do navegador é que a página inicial da Owasp Broken será acessada, na qual estarão os acessos aos sistemas a serem testados.

A figura 19 mostra como os testes e vulnerabilidades cobertos por este guia são inseridos no processo de teste adotado, conforme a metodologia da Owasp.

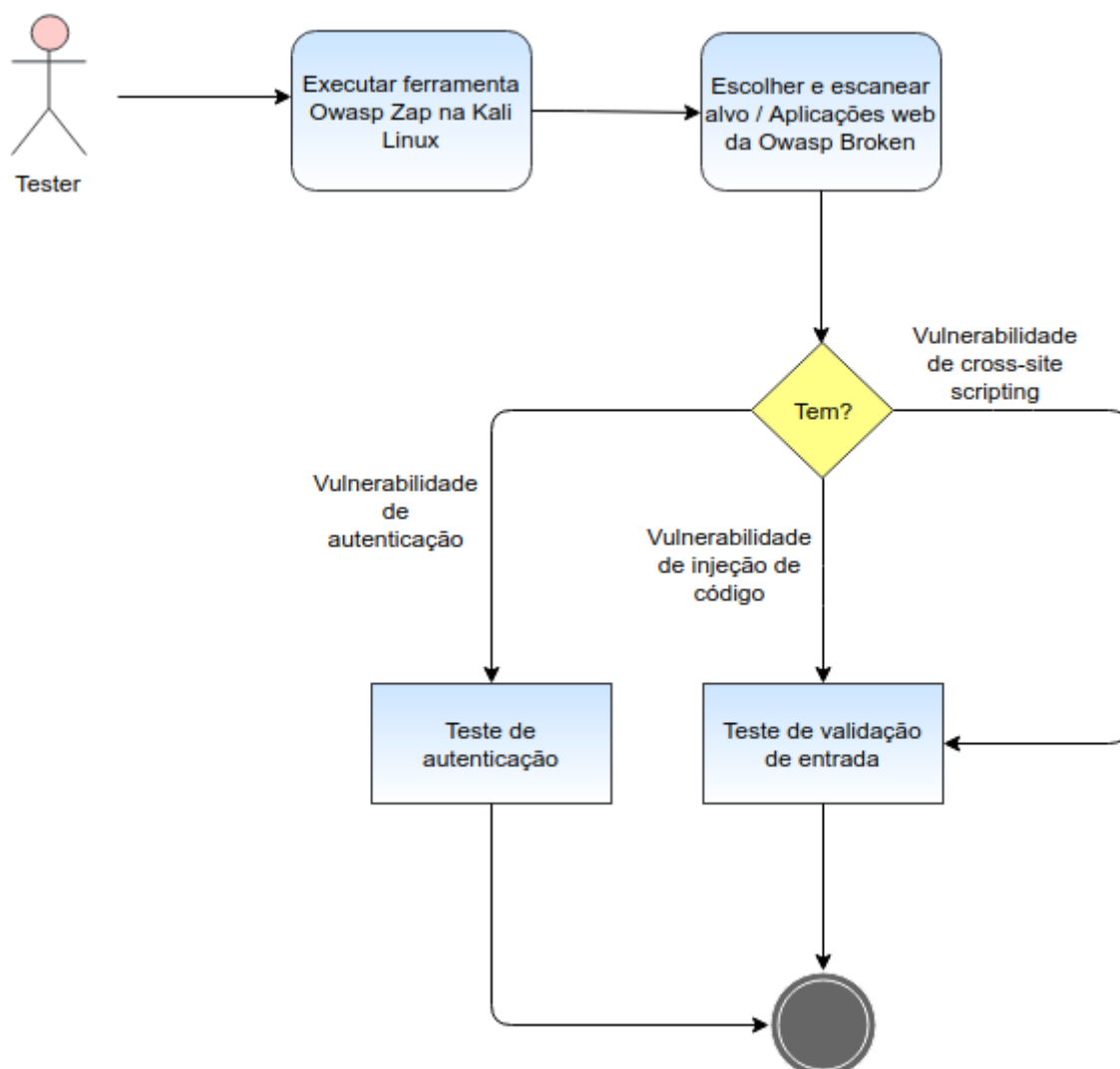


Figura 19 - Diagrama de execução do desenvolvimento

4.2 CONFIGURAÇÃO DO AMBIENTE

Para configurar o ambiente no qual serão realizados os testes de intrusão é preciso de duas máquinas virtuais, uma contendo o sistema operacional Kali Linux e outra com a máquina da Owasp Broken. O software que será utilizado para a criação das máquinas virtuais é o VirtualBox.

A imagem da Kali Linux pode ser encontrada no link <https://www.offensive-security.com/kali-linux-vmware-virtualbox-image-download> com a descrição de Kali Linux 64 bit VBox e seu tamanho é de 3.1 GB, conforme a figura 20. Também é possível fazer o download de uma imagem na versão de 32 bit ou uma versão mais leve.

Image Name	Torrent	Size	Version	SHA256Sum
Kali Linux 64 bit VBox	Torrent	3.1G	2017.1	9c1144090971ede73937ee6266013054252bfff19b306ae8ec8b55f08249c1fcc
Kali Linux 32 bit VBox PAE	Torrent	3.1G	2017.1	340bebd610f84c148077df4780b7e8d6736802bd4c857a59ace8fce79bb2ce42
Kali Linux Light 64 bit VBox	Torrent	0.8G	2017.1	698bb3d4e680b54a41aea38a1f8afd3088d4b01dfec80d5ba8400f6f84679d8f
Kali Linux Light 32 bit VBox	Torrent	0.8G	2017.1	8b1af1ff0c81567342367e11d28fde79016340823837a8d4f339a21e91c516c0

Figura 20 - Imagens de download Kali Linux

A imagem da máquina Owasp Broken pode ser encontrada no link <https://sourceforge.net/projects/owaspbwa/files/1.2>. A versão utilizada neste guia é a 1.2 com tamanho de 2.6 GB. Outros arquivos podem ser encontrados no mesmo link, onde a imagem estará compactada.

Caso as máquinas virtuais sejam importadas no VirtualBox instalado num sistema linux, basta abrir o arquivo de extensão .ova. Feito isso, uma tela semelhante a da figura 21 mostra as configurações já definidas para a máquina a ser importada. Basta apertar o botão de importar e aguardar o processo terminar. É necessário realizar esse procedimento para as duas máquinas virtuais. A figura 22

mostra como deve estar o estado do VirtualBox após importar ambas as máquinas virtuais.



Figura 21 - Importação de máquina virtual

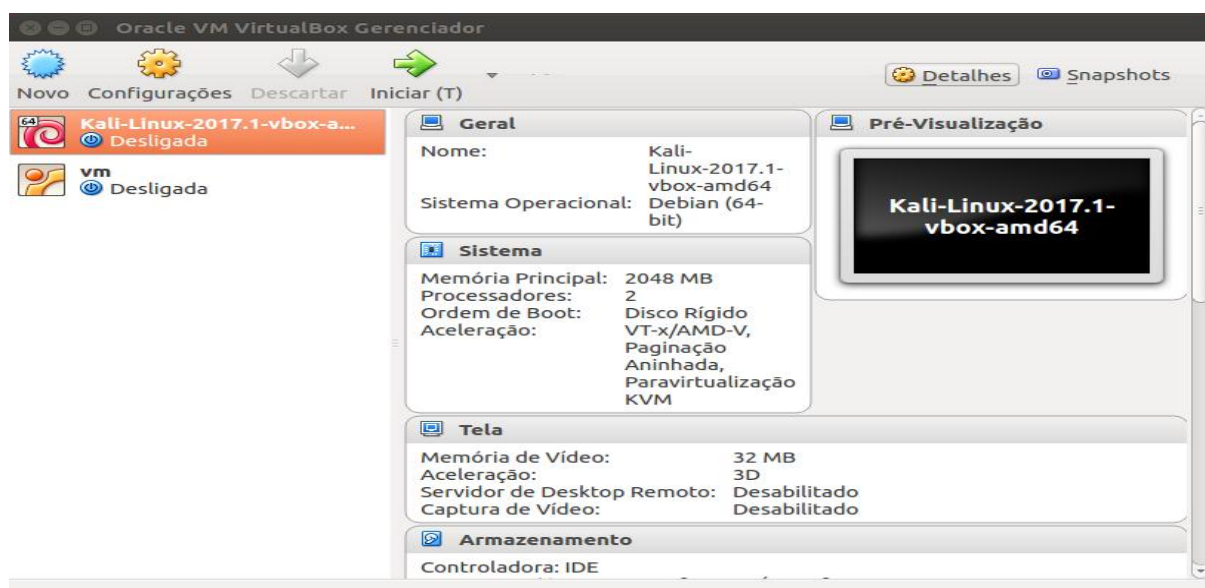


Figura 22 - Máquinas virtuais importadas

Após importar as máquinas é necessário configurar uma rede no VirtualBox que permita a elas se comunicarem e ter acesso a internet. O primeiro passo a fazer é configurar o servidor DHCP, que será responsável por distribuir e alocar os endereços ip para cada nodo da rede e para isso deve-se na tela do VirtualBox apertar as teclas CTRL+G ou ir em Arquivo / Preferências. Abrirá uma tela conforme a figura 23, na qual é necessário clicar na opção Rede e na guia Redes Exclusivas de Hospedeiro.

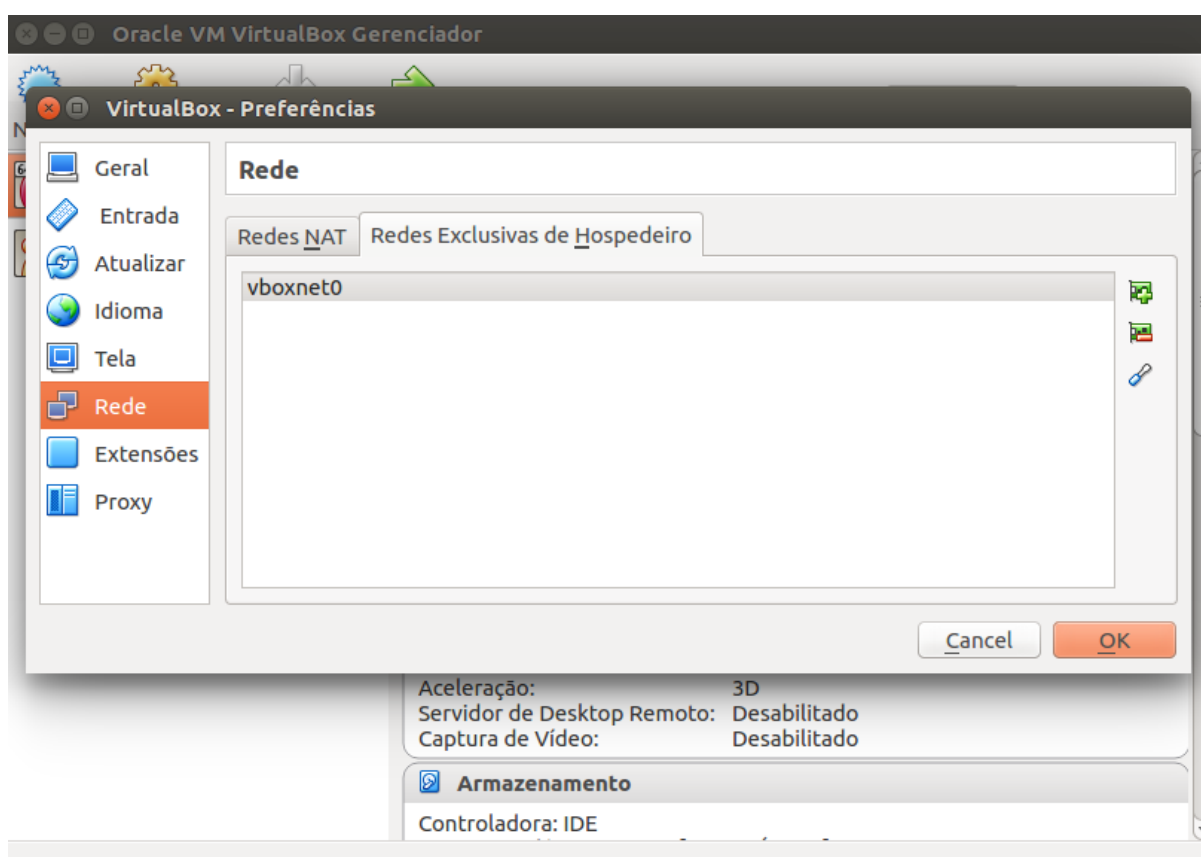


Figura 23 - Redes Exclusivas de Hospedeiro

Deve-se editar a placa de rede que já vem configurada ou então se não vier, conforme a figura 23, adicionar uma. Para editar, com o botão direito do mouse

selecionar a opção Editar Rede Exclusiva de Hospedeiro e configurar a tela que abrir com os valores presentes nas figuras 24 e 25.

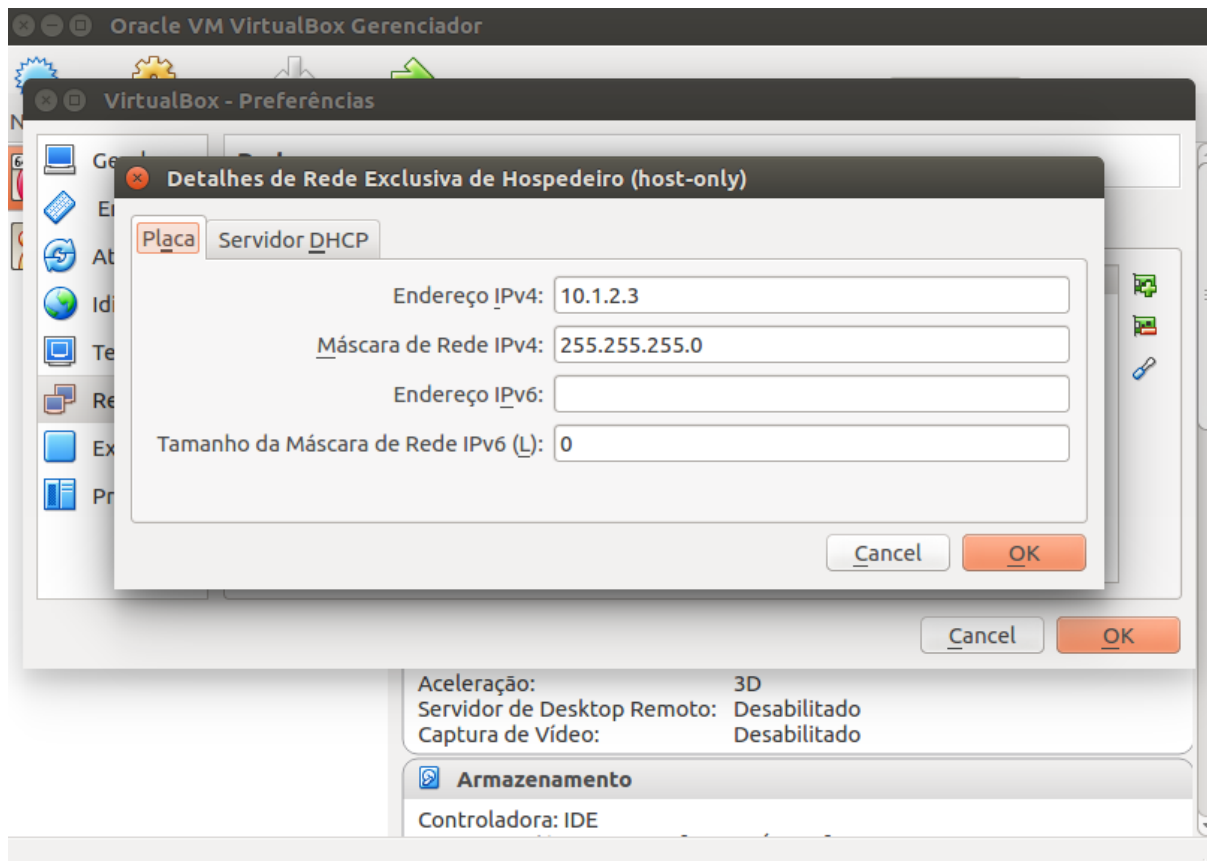


Figura 24 - Endereço ip da placa virtual

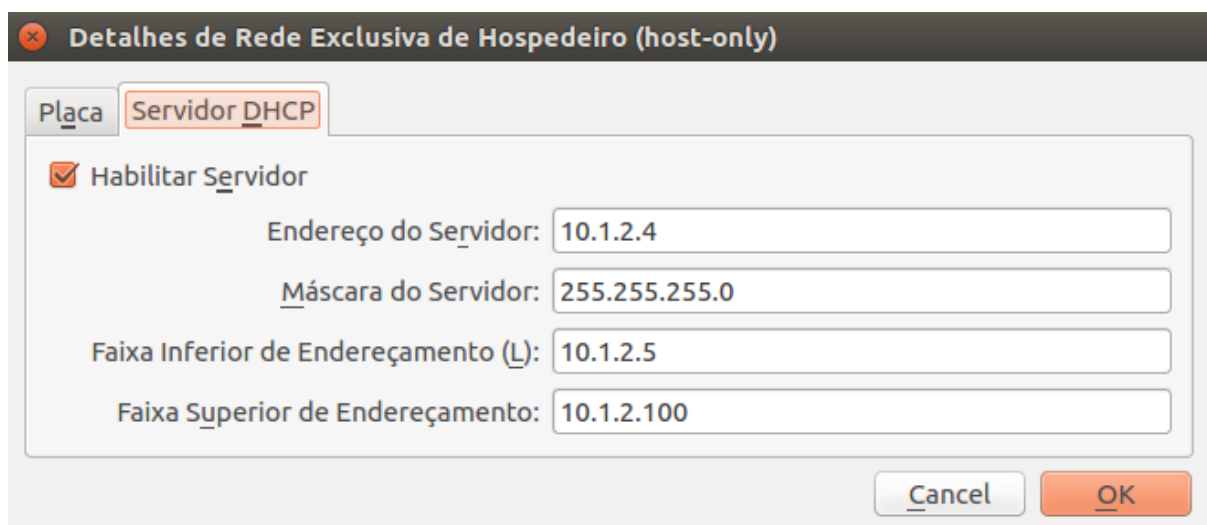


Figura 25 - Servidor DHCP

Agora é preciso configurar a placa de rede de cada máquina virtual. Começando pela Kali Linux deve-se selecioná-la e clicar em configurações na tela do VirtualBox. Selecionando a opção Rede, é preciso configurar o Adaptador 1 como Placa de rede exclusiva de hospedeiro, conforme a figura 26.

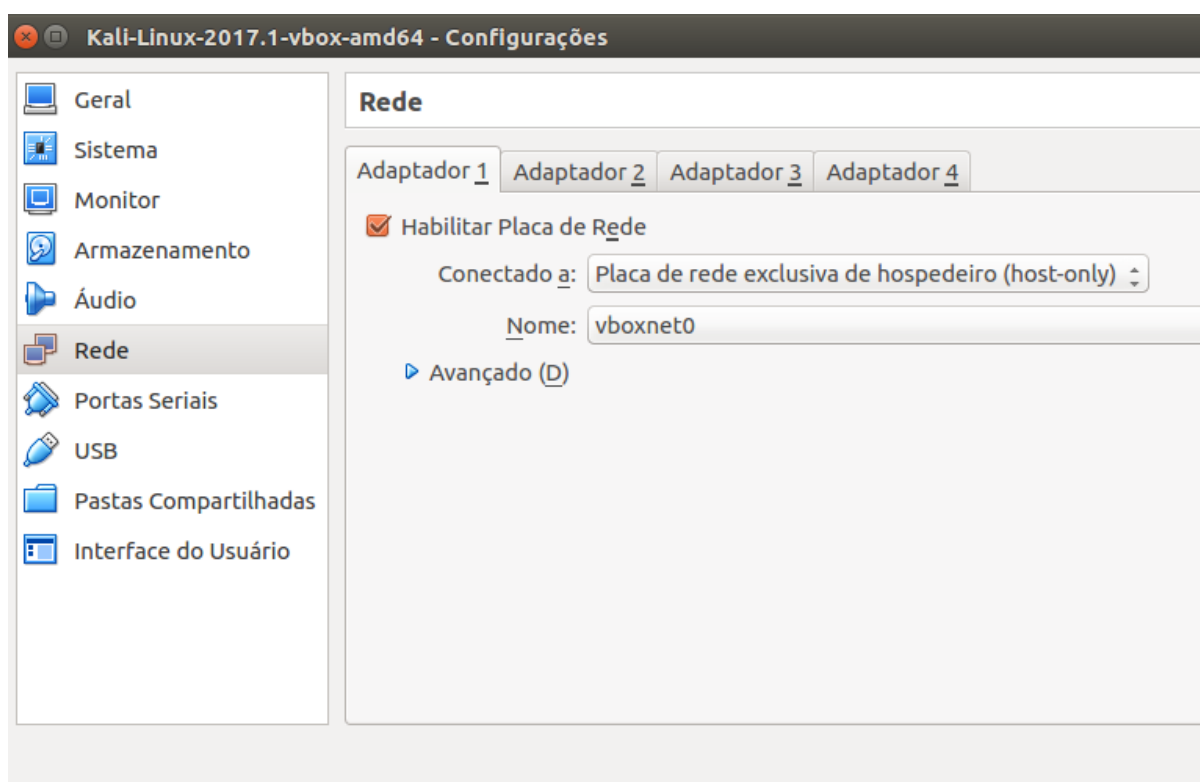


Figura 26 - Adaptador 1 da máquina Kali Linux

Na aba do Adaptador 2 será configurada uma placa como NAT e é ela quem traduz os endereços de ip de cada máquina para ter acesso a rede externa e a Internet. A configuração deve ser feita conforme a figura 27.

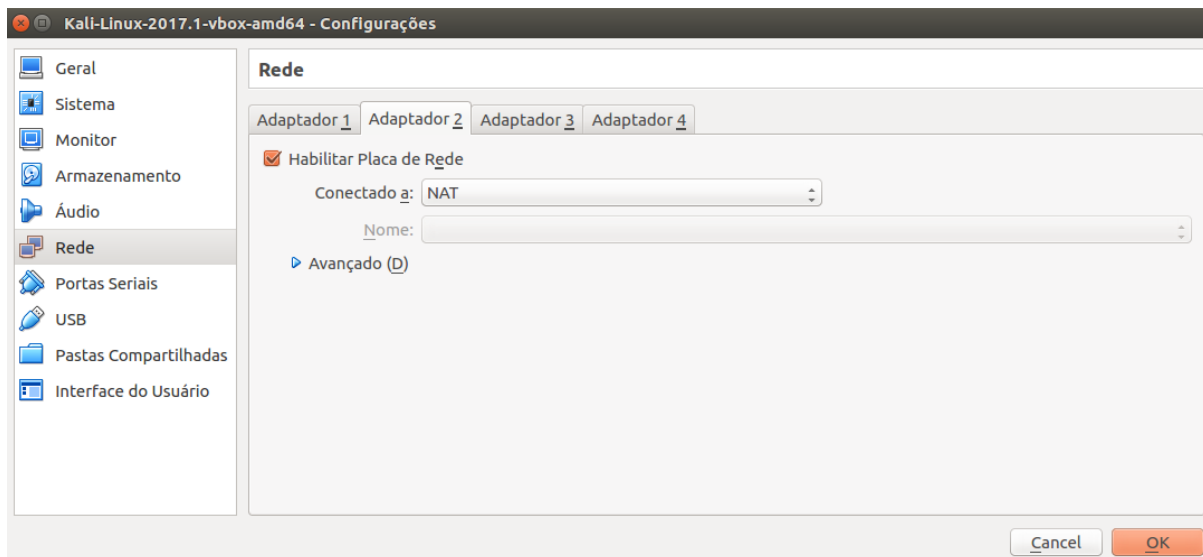


Figura 27 - Configuração da placa NAT

Após configurar as placas na máquina Kali Linux, será configurada uma placa de rede exclusiva de hospedeiro na Owasp Broken, conforme a figura 28.

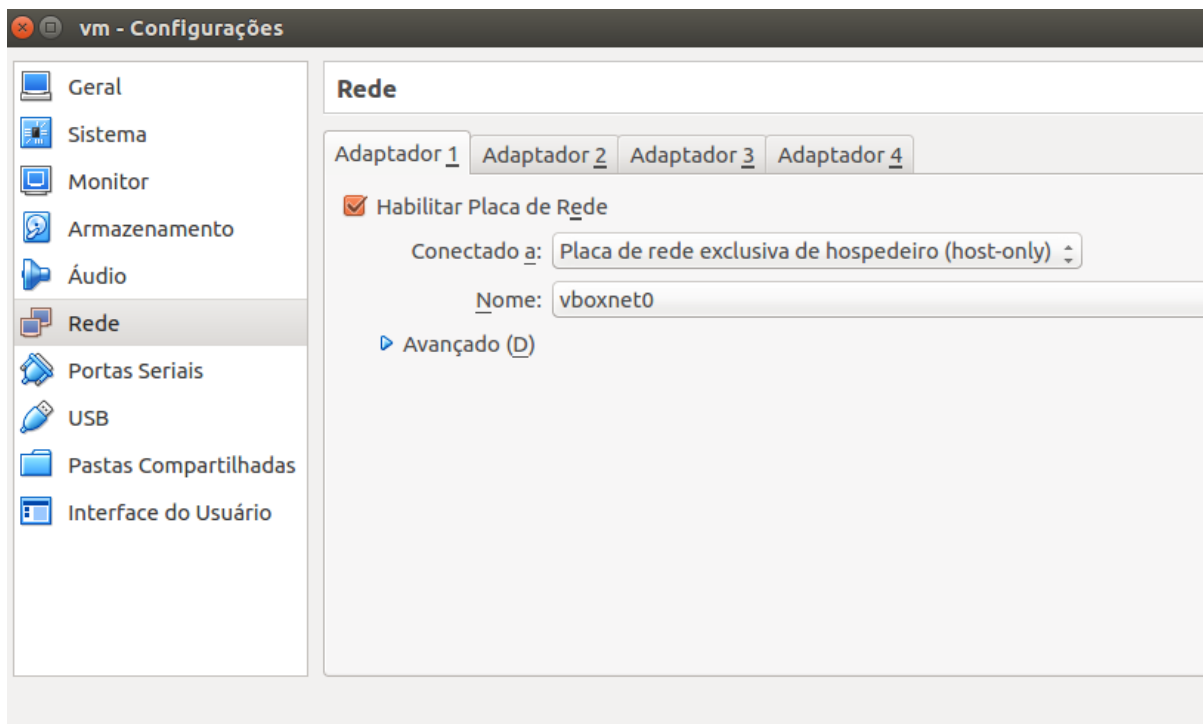
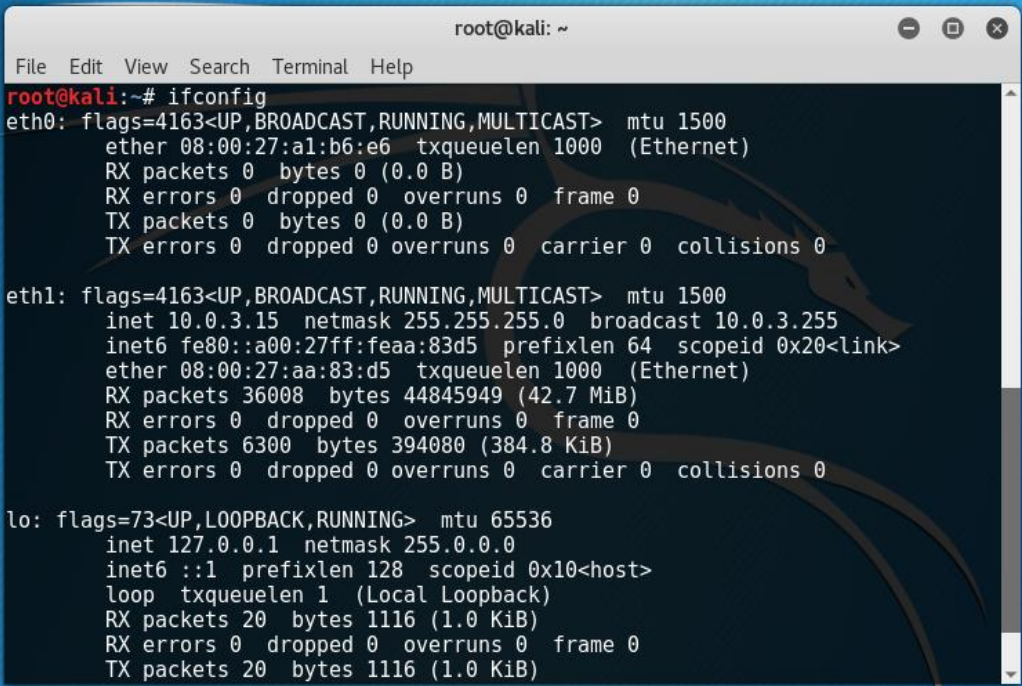


Figura 28 - Adaptador 1 da máquina Owasp Broken

Com essas configurações as máquinas virtuais importadas já conseguem ter acesso a Internet caso o computador em que o VirtualBox esteja rodando também tenha acesso. Sendo assim, para realizar o último passo para terminar a configuração do ambiente que será utilizado para fazer os testes, é necessário iniciar a máquina Kali Linux.

Toda vez que a Kali for inicializada ela solicitará um usuário e senha que darão acesso ao sistema, o usuário é root e a senha é toor. Uma vez realizado o *login* no sistema, precisa-se verificar se todas as interfaces de rede estão devidamente conectadas. Para isso é preciso abrir o terminal da Kali e conforme a figura 29 digitar o comando `ifconfig` para listar as interfaces disponíveis.



```
root@kali: ~  
File Edit View Search Terminal Help  
root@kali:~# ifconfig  
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500  
    ether 08:00:27:a1:b6:e6 txqueuelen 1000  (Ethernet)  
    RX packets 0  bytes 0 (0.0 B)  
    RX errors 0  dropped 0  overruns 0  frame 0  
    TX packets 0  bytes 0 (0.0 B)  
    TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0  
  
eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500  
    inet 10.0.3.15 netmask 255.255.255.0 broadcast 10.0.3.255  
    inet6 fe80::a00:27ff:feaa:83d5 prefixlen 64 scopeid 0x20<link>  
    ether 08:00:27:aa:83:d5 txqueuelen 1000  (Ethernet)  
    RX packets 36008  bytes 44845949 (42.7 MiB)  
    RX errors 0  dropped 0  overruns 0  frame 0  
    TX packets 6300  bytes 394080 (384.8 KiB)  
    TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0  
  
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536  
    inet 127.0.0.1 netmask 255.0.0.0  
    inet6 ::1 prefixlen 128 scopeid 0x10<host>  
    loop txqueuelen 1  (Local Loopback)  
    RX packets 20  bytes 1116 (1.0 KiB)  
    RX errors 0  dropped 0  overruns 0  frame 0  
    TX packets 20  bytes 1116 (1.0 KiB)
```

Figura 29 - Interfaces de rede da Kali Linux

Observando as interfaces listadas na figura 29, pode-se perceber que a interface eth0 não obteve nenhum endereço de ip. Através do menu superior direito, conforme a figura 30, é possível ver que a interface eth0 está desligada.

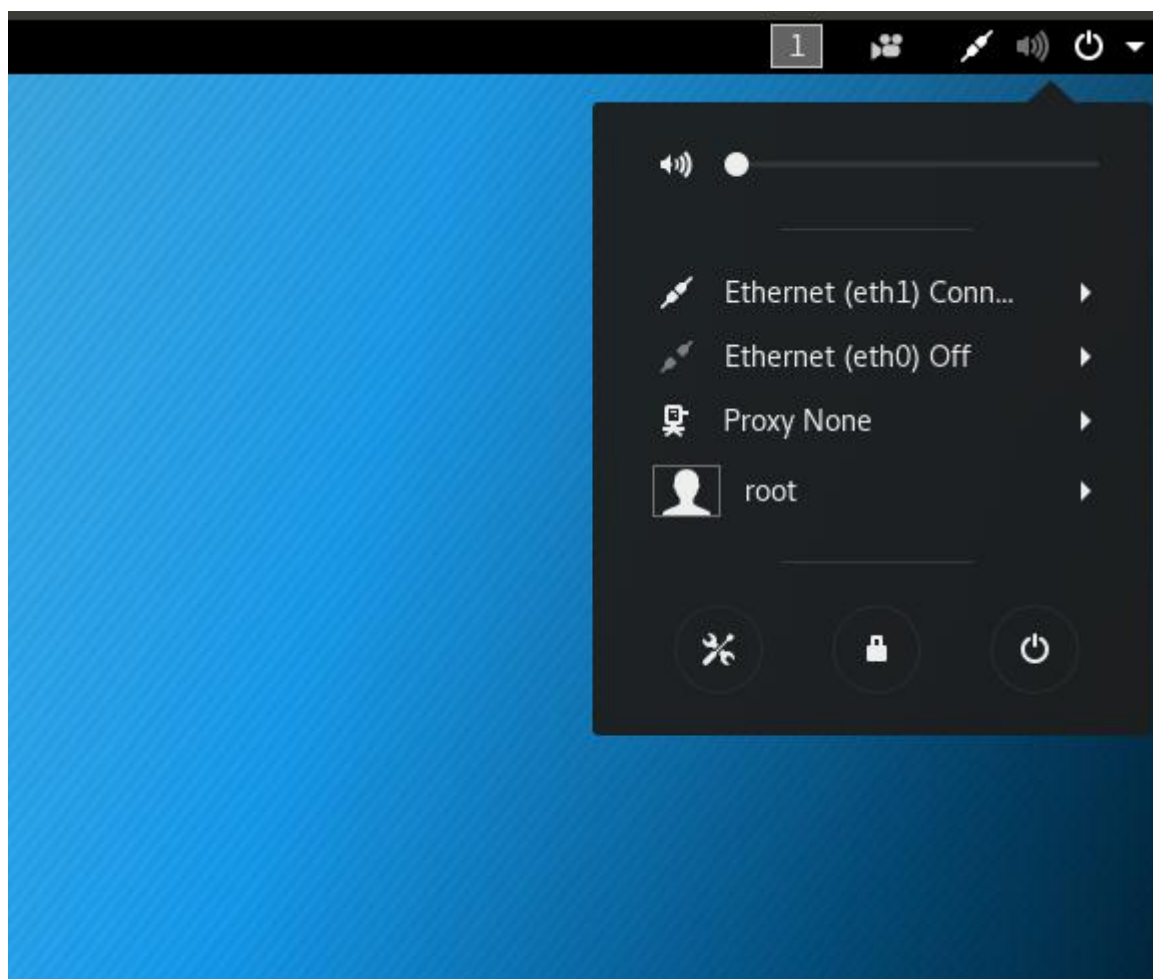


Figura 30 - Placas de rede na interface gráfica da Kali Linux

Através deste mesmo menu deve-se ligar a placa de rede. Basta clicar nela e posteriormente na opção *Wired Settings* que estará disponível, abrindo uma tela de configuração das placas de rede. Clique no botão ao lado direito da tela para ativar a interface de rede e em seguida em *Add Profile*.

Na tela que abrir, basta clicar em Add e as placas de rede agora devem estar conectadas e com o perfil de Profile 1, como na figura 31. A opção de *Add Profile* é utilizada para gerar um perfil de configuração e permitir que toda vez ao ligar a máquina Kali Linux ambas as interfaces de rede venham conectadas.

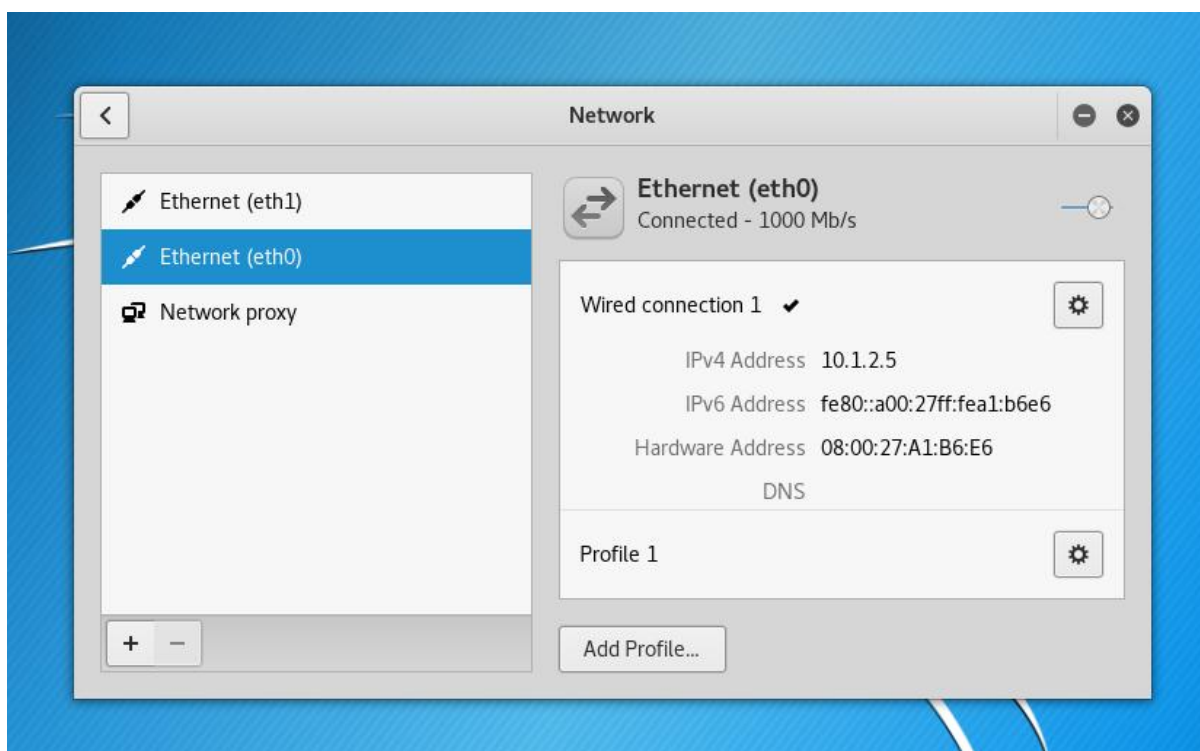
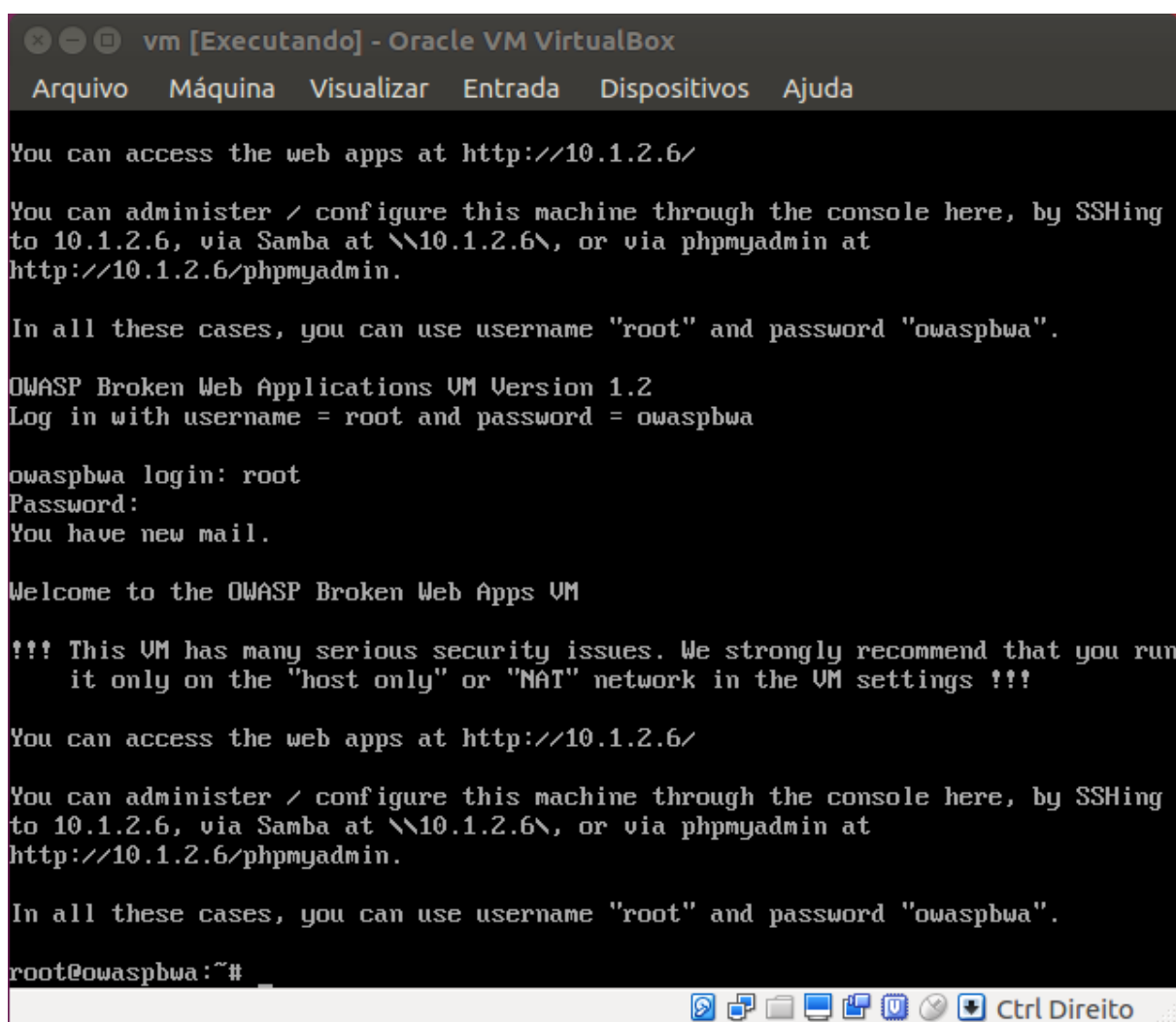


Figura 31 - Placas de rede ativas e com Profile 1

Para verificar se as interfaces de rede agora estão cada uma com um endereço de ip, basta digitar novamente o comando `ifconfig` no terminal. Já é possível iniciar a máquina Owasp Broken e ela também solicitará um usuário e uma senha, o usuário é `root` e a senha é `owaspbwa`.

A máquina Broken basicamente roda localmente num servidor web um conjunto de aplicações web vulneráveis para serem testadas. Ela não apresentará

interface gráfica, somente exibirá um terminal contendo algumas informações para auxiliar quem a utilizará, incluindo o ip local em que ela subiu as aplicações. Pode-se observar isso na figura 32. Com nossas configurações a máquina Kali Linux deve ter uma interface de rede com o ip 10.1.2.5 e o acesso às aplicações web na Owasp Broken deve ser realizado por meio do ip 10.1.2.6.



```
vm [Executando] - Oracle VM VirtualBox
Arquivo  Máquina  Visualizar  Entrada  Dispositivos  Ajuda

You can access the web apps at http://10.1.2.6/

You can administer / configure this machine through the console here, by SSHing
to 10.1.2.6, via Samba at \\10.1.2.6\\, or via phpmyadmin at
http://10.1.2.6/phpmyadmin.

In all these cases, you can use username "root" and password "owaspbwa".

OWASP Broken Web Applications VM Version 1.2
Log in with username = root and password = owaspbwa

owaspbwa login: root
Password:
You have new mail.

Welcome to the OWASP Broken Web Apps VM

!!! This VM has many serious security issues. We strongly recommend that you run
it only on the "host only" or "NAT" network in the VM settings !!!

You can access the web apps at http://10.1.2.6/

You can administer / configure this machine through the console here, by SSHing
to 10.1.2.6, via Samba at \\10.1.2.6\\, or via phpmyadmin at
http://10.1.2.6/phpmyadmin.

In all these cases, you can use username "root" and password "owaspbwa".

root@owaspbwa:~#
```

Figura 32 - Inicialização da máquina virtual Owasp Broken

Antes de finalizar o processo de configuração, deve-se configurar a ferramenta Owasp Zap para estar pronta quando começar a realizar os testes de intrusão. Na máquina Kali Linux para abrir a ferramenta basta seguir como a figura 33.

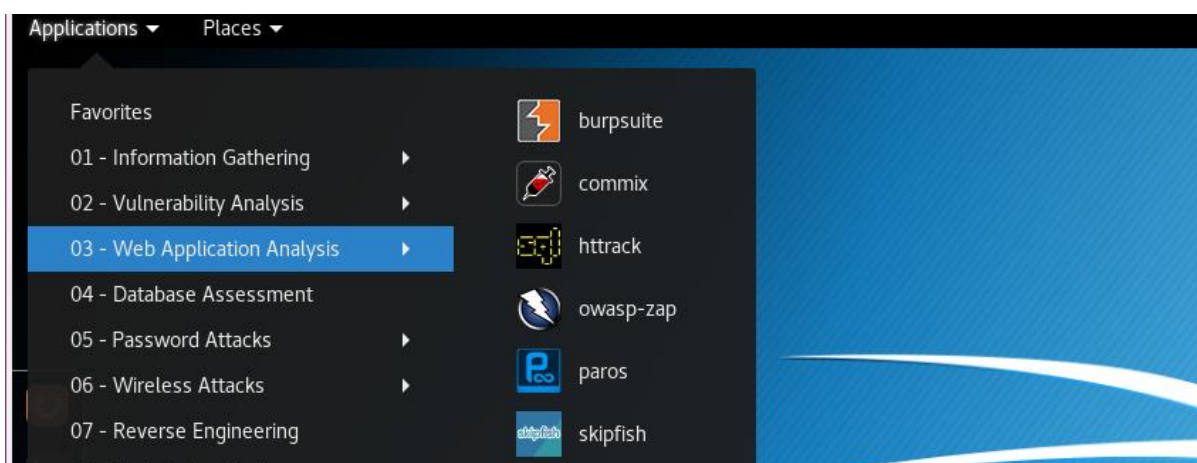


Figura 33 - Abrindo a ferramenta Owasp Zap

Na primeira vez ao abrir a Owasp Zap será solicitado para aceitar a licença e posteriormente se deseja salvar a sessão da ferramenta, para essa segunda opção deve-se escolher não e marcar para lembrar a escolha. Ao abrir a ferramenta é possível alterar o idioma dela para português, indo através do menu em *tools* e depois *options* e *language*.

Com a tela principal da ferramenta aberta é possível ver alguns recursos já disponíveis. No centro da tela da figura 34 está o campo URL para atacar, este é o campo mais básico e rápido de todos para um usuário iniciante, basta digitar nele o link da aplicação ou site no qual se deseja encontrar vulnerabilidades e clicar em

Ataque, disparando o conjunto de utilitários que a ferramenta Owasp Zap tem para encontrar vulnerabilidades.

Logo após disparar um ataque é possível ver abrir algumas abas na parte inferior da tela. As abas que mais são utilizadas neste guia são a aba de Alertas e Varredura Ativa. Em Alertas é onde aparecem quais problemas e vulnerabilidades foram encontradas na aplicação escaneada, separando cada uma. Varredura Ativa é onde estão os ataques de teste realizados para encontrar as vulnerabilidades.

Ao lado esquerdo da ferramenta é possível ver um painel com uma lista chamada *Sites*, na qual estarão o conjunto de urls visitadas, permitindo escanear caminhos específicos de uma aplicação.

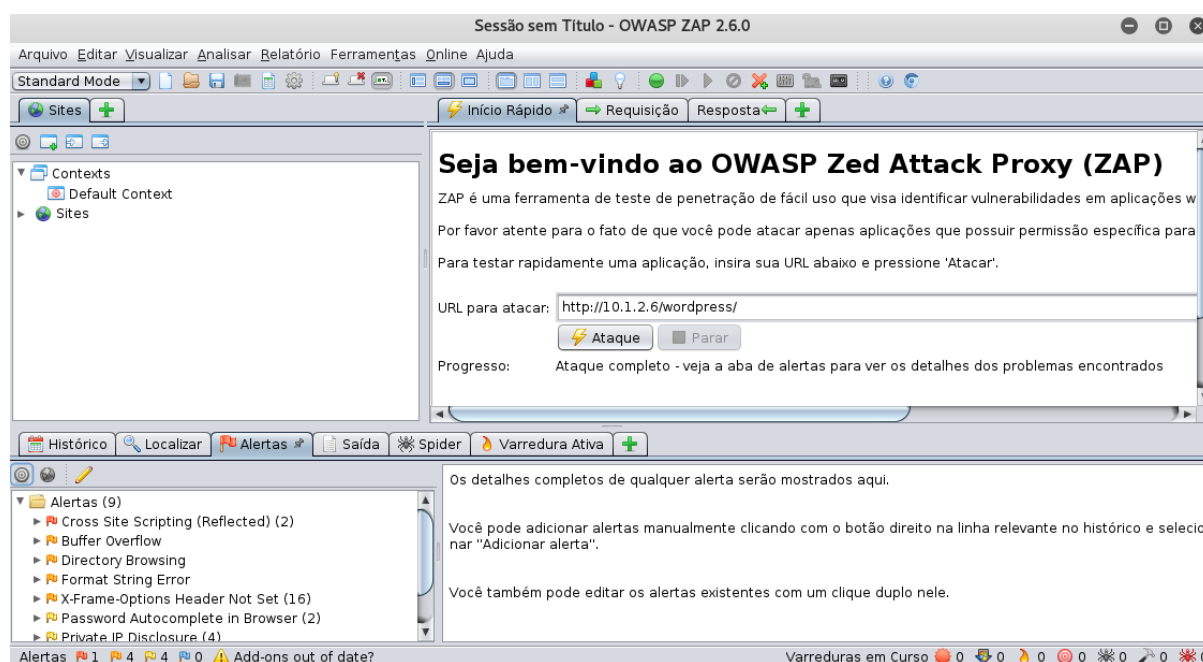


Figura 34 - Básico da Owasp Zap

Para finalizar o processo de configuração do ambiente é preciso configurar uma das principais funções da ferramenta Zap e o navegador no qual abriremos as aplicações.

Além de permitir varrer aplicações em busca de vulnerabilidades, a Owasp Zap também pode ser configurada para ficar como *proxy* entre o servidor da aplicação e o navegador (cliente). O *proxy* basicamente recebe a requisição vinda do navegador através da interação com a aplicação do lado cliente e repassa essa requisição ao servidor. A resposta vinda do servidor também é repassada primeiramente ao *proxy* para depois voltar ao cliente, permitindo ao tester de segurança tanto verificar o histórico de requisições e respostas, como debugar isoladamente cada uma. Também é possível visualizar os parâmetros das requisições e realizar uma análise mais manual das requisições disparadas ao servidor e as respostas. Isso é o que basicamente será utilizado para descobrir as vulnerabilidades nas aplicações a serem testadas.

Para configurar o *proxy* na Zap, deve-se acessar o menu ferramentas e opções. Ao abrir a nova tela, a opção *Proxy Local*, deverá estar conforme a figura 35. O endereço ip colocado nesta tela deverá ser o mesmo que será colocado no navegador, pois é através deste endereço e porta que o *proxy* ficará interceptando as requisições e respostas.

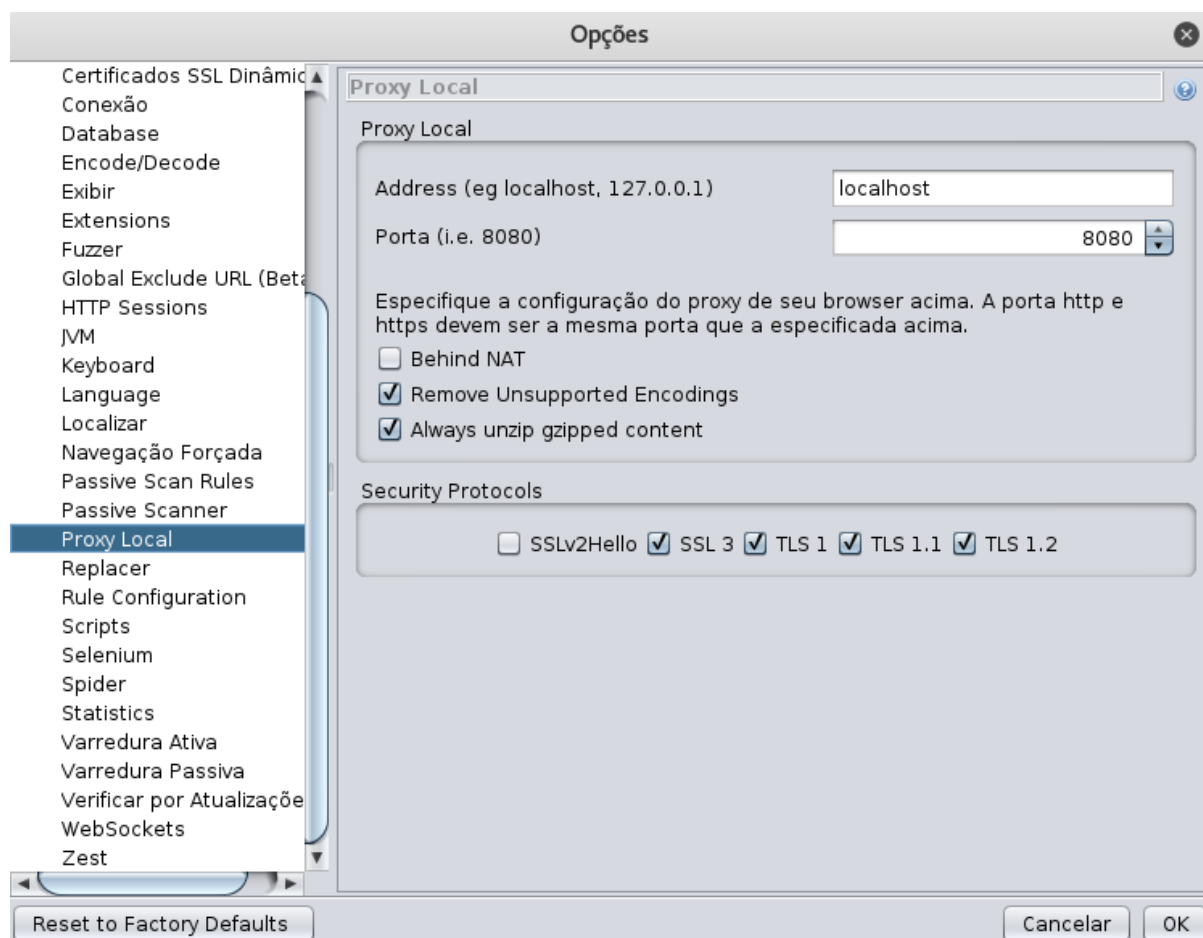


Figura 35 - Configuração do *proxy* na OJAP

Se as aplicações a serem testadas fazem uso do protocolo SSL é necessário gerar um certificado digital através da Zap e importar ele no navegador onde a aplicação será testada, para que a Zap possa interceptar requisições que usam esse protocolo. Para isso nesse mesmo menu há a opção Certificados SSL Dinâmicos, que permite salvar um certificado como mostrado na figura 36. Lembre-se de salvar o certificado num local fácil, pois será importado no navegador na máquina Kali Linux. Clique em *OK* e a ferramenta já estará totalmente configurada para os testes.

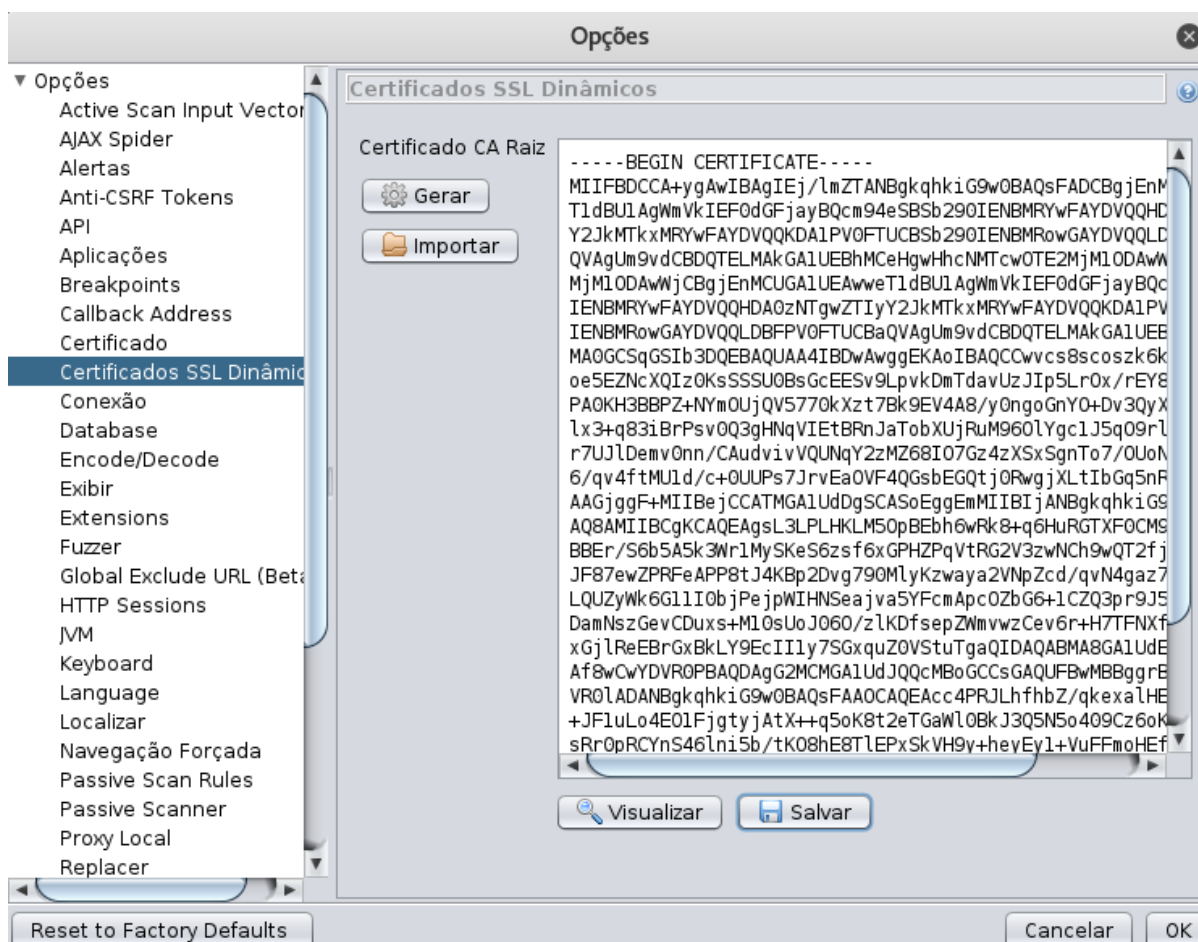


Figura 36 - Certificado SSL

Para realizar os testes será utilizado, na Kali Linux, o navegador Firefox. Ele pode ser aberto através do menu lateral, conforme a figura 37. Depois de aberto, o navegador deve ser configurado para que toda requisição que ele faça, passe primeiro pelo *proxy* da Zap. Para isso basta acessar o menu de preferências do navegador mostrado na figura 38.

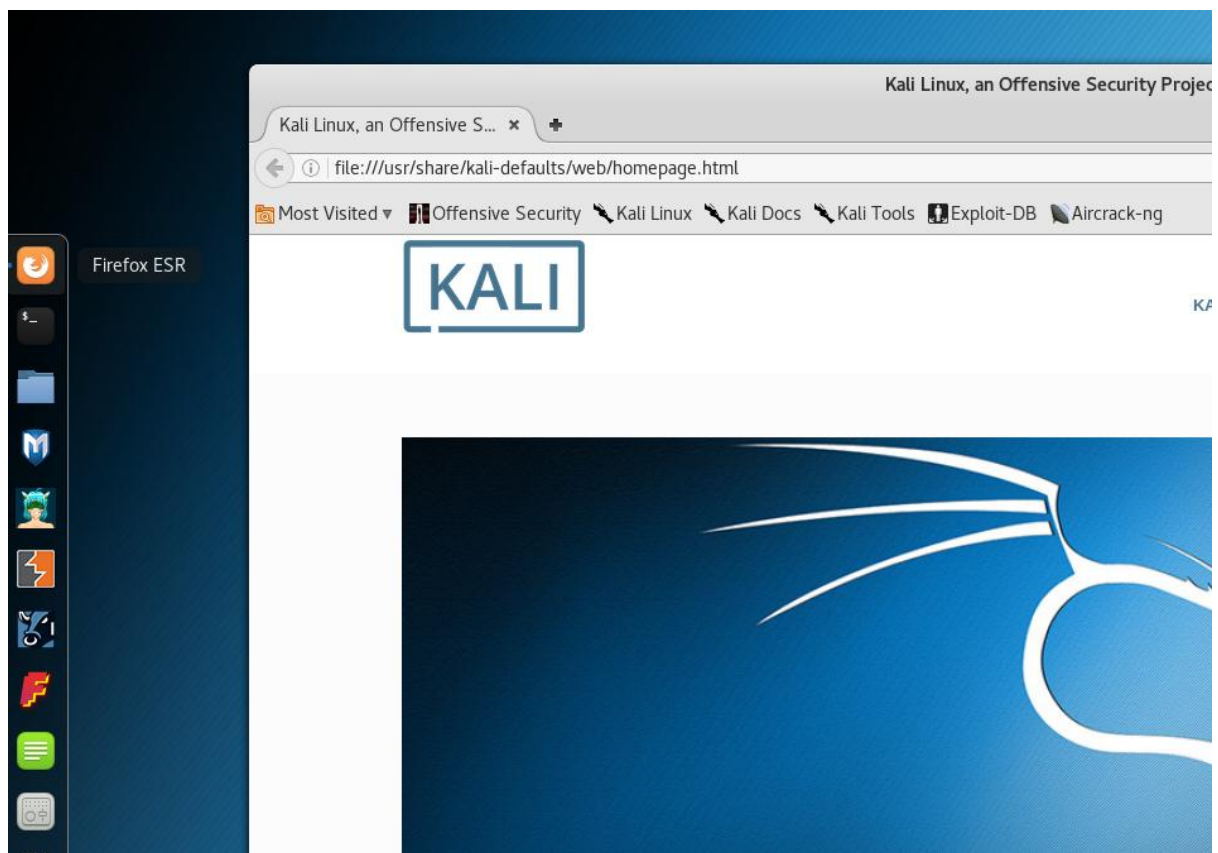


Figura 37 - Abrir navegador Firefox

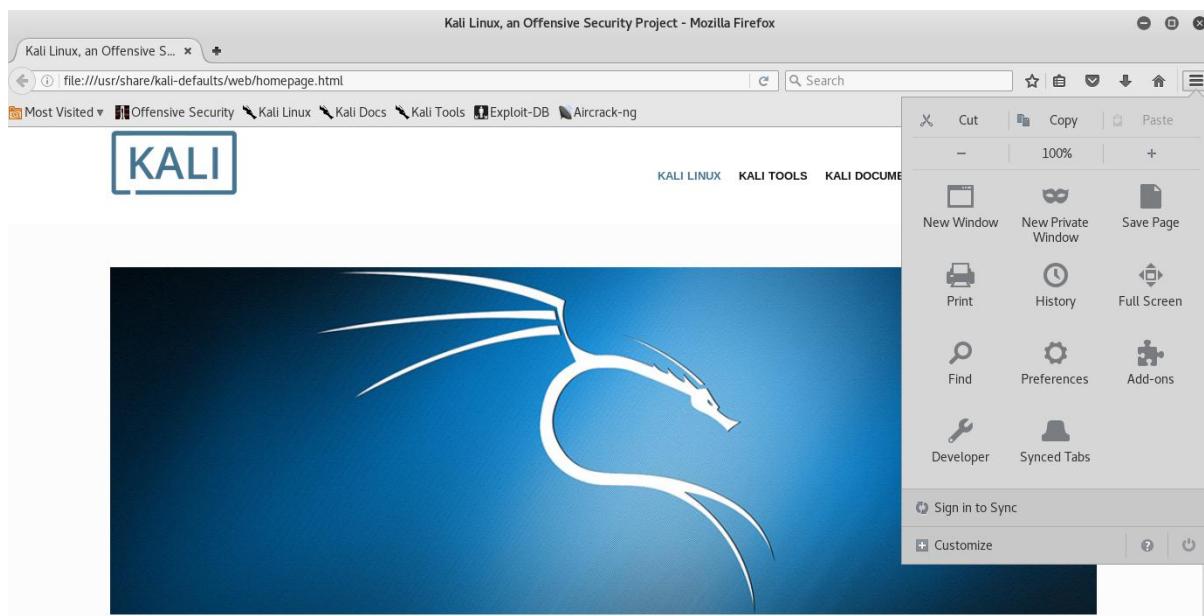


Figura 38 – Preferences

Ao acessar as preferências do navegador é preciso abrir a aba *Advanced* e então *Network*. Na tela que estiver é só clicar em *Settings* da seção *Connection* e configurar conforme a figura 39, e então clicar em *OK*. Com isso o ambiente já deve estar todo configurado e pronto para realizar os testes.

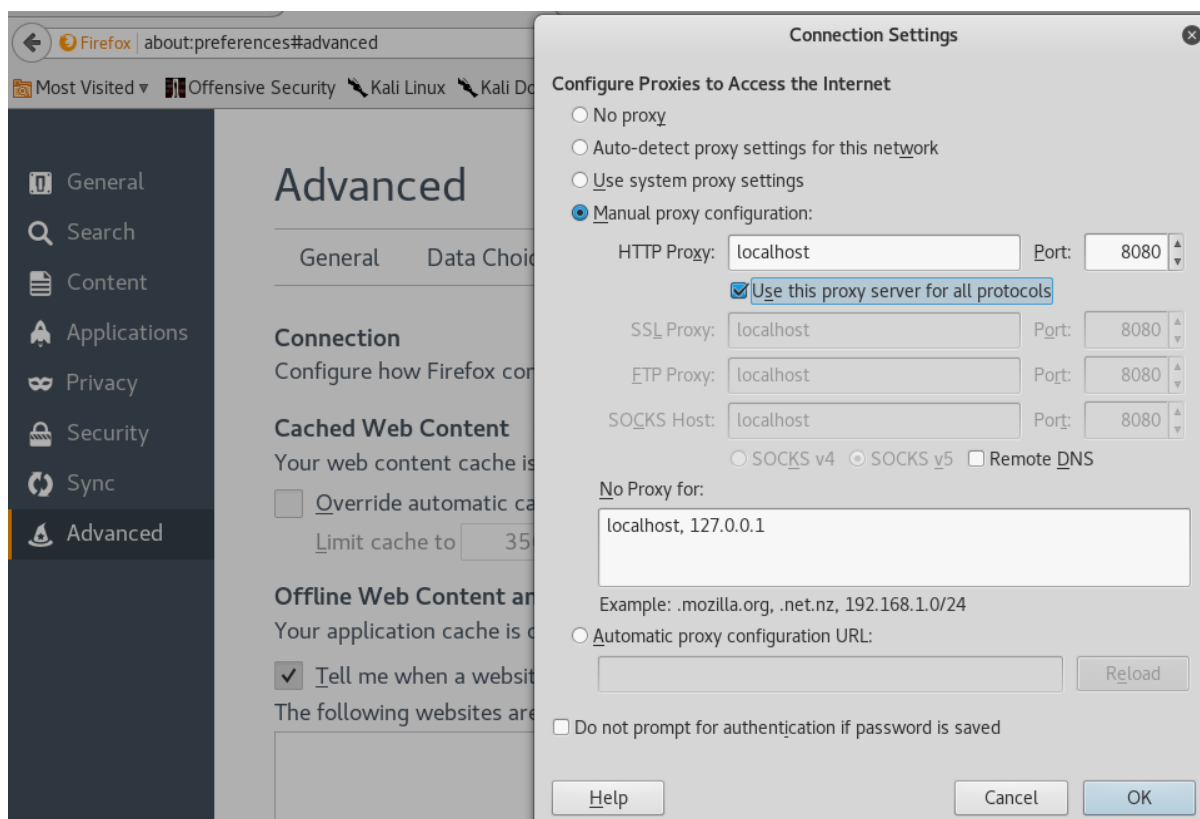


Figura 39 - Configuração do *proxy* no navegador

4.3 EXECUÇÃO DOS TESTES DE VALIDAÇÃO DE ENTRADA

4.3.1 Teste de *Cross-Site Scripting* Refletido

Como descrito na seção de segurança de aplicações web, o ataque de *cross-site scripting* refletido ocorre quando um usuário executa um *link* que apresenta uma

string de ataque e dentro da resposta *HTTP* o código é retornado ao usuário e executado em seu navegador.

Segundo a OWASP (2014), o processo de construção de um ataque de XSS refletido apresenta algumas etapas. Primeiro a etapa de arquitetura do ataque, na qual o atacante cria e testa uma determinada URL contendo código malicioso. Etapa de engenharia social em que tenta-se convencer a vítima de executar o *link* em seu navegador e possivelmente uma etapa de exploração de acesso ao computador no qual o código malicioso foi executado.

Para realizar esse teste e verificar se uma aplicação web está vulnerável a *cross-site scripting* refletido são incluídas três fases, conforme OWASP (2014).

A primeira fase é a detecção dos pontos de entrada de dados da aplicação. Deve-se verificar cada página em busca de campos de entrada em formulários, parâmetros das requisições HTTP, dados que são enviados no corpo das requisições e até campos escondidos ou com valores pré-definidos. Para analisar as requisições em busca de parâmetros que possam ser utilizados como entrada de dados é possível utilizar a ferramenta Owasp Zap configurada na seção do ambiente.

A segunda fase trata a análise das entradas de dados levantadas na fase anterior. Para cada entrada é possível verificar se ela é vulnerável ou não utilizando dados especialmente criados com o intuito de executar comandos no navegador. Nessa etapa podem ser utilizadas ferramentas que possuem listas de *strings* de ataque definidas, a própria Owasp Zap faz *scan* dessas entradas tentando detectar se estão vulneráveis.

A terceira e última etapa deste teste é verificar o resultado dos testes da fase anterior e determinar se para os campos analisados existe alguma vulnerabilidade de real impacto para a aplicação. Deve-se examinar as páginas web resultantes e verificar os campos testados para detectar se algum *script* foi executado ou se a aplicação não filtrou corretamente os campos.

Execução do teste na aplicação WordPress

Para demonstrar o teste de *cross-site scripting* refletido foi escolhida a aplicação WordPress em sua versão 2.0.0, presente na máquina da Owasp Broken. WordPress é uma plataforma para publicação de conteúdo na Internet, mundialmente conhecida e que em versões antigas já apresentou vulnerabilidades.

As máquinas Owasp Broken e Kali Linux devem estar ligadas para a continuação do teste. Como a ferramenta Owasp Zap foi configurada como proxy na máquina Kali é necessário que ela esteja aberta para se ter acesso às aplicações que serão testadas a partir da Kali.

Para acessar a aplicação WordPress é necessário abrir o navegador Firefox, presente na máquina Kali e já configurado. No *browser* ao acessar o endereço 10.1.2.6 todas as aplicações presentes na máquina Owasp Broken estão disponíveis, conforme a figura 40.

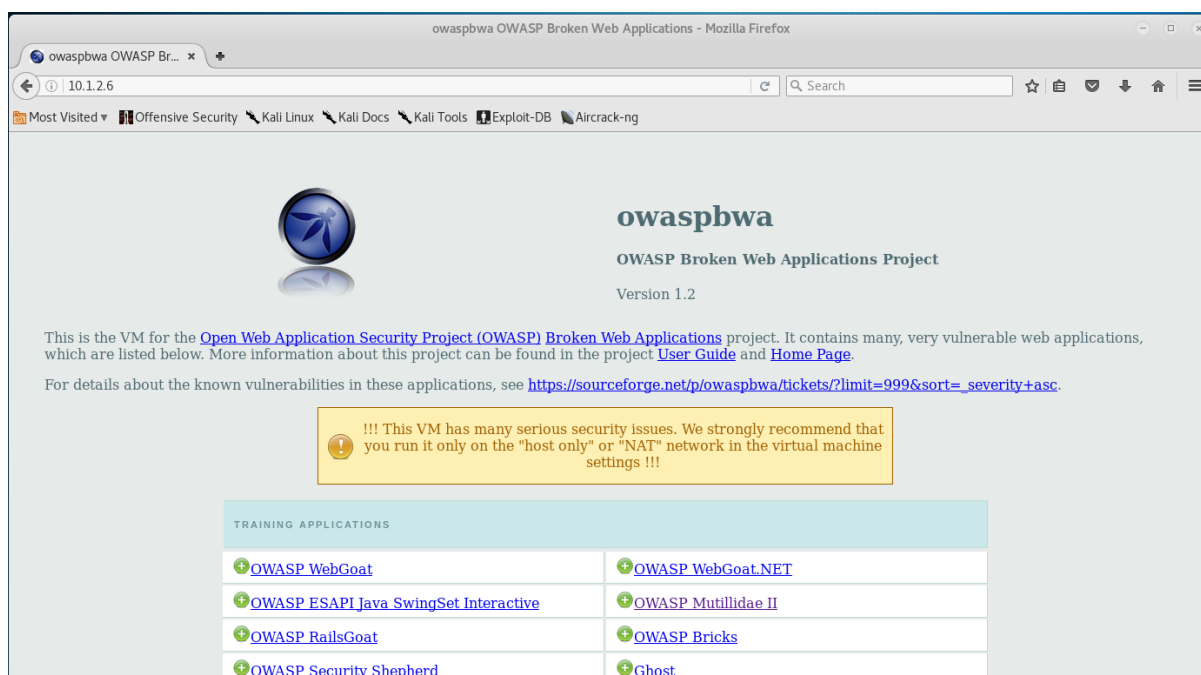


Figura 40 - Aplicações da máquina Owasp Broken

Na página aberta procurar pelo *link* WordPress e clicar para acessar a aplicação. Como primeira etapa desse teste, a aplicação é explorada em busca de campos de entrada de dados.

Ao navegar na tela inicial do WordPress é possível ver *links*, comentários e até um campo de busca que poderia também ser testado. Na mesma tela existe um *link* chamado *Register*, presente na figura 41, que dá acesso à uma página de registro de usuário na aplicação.

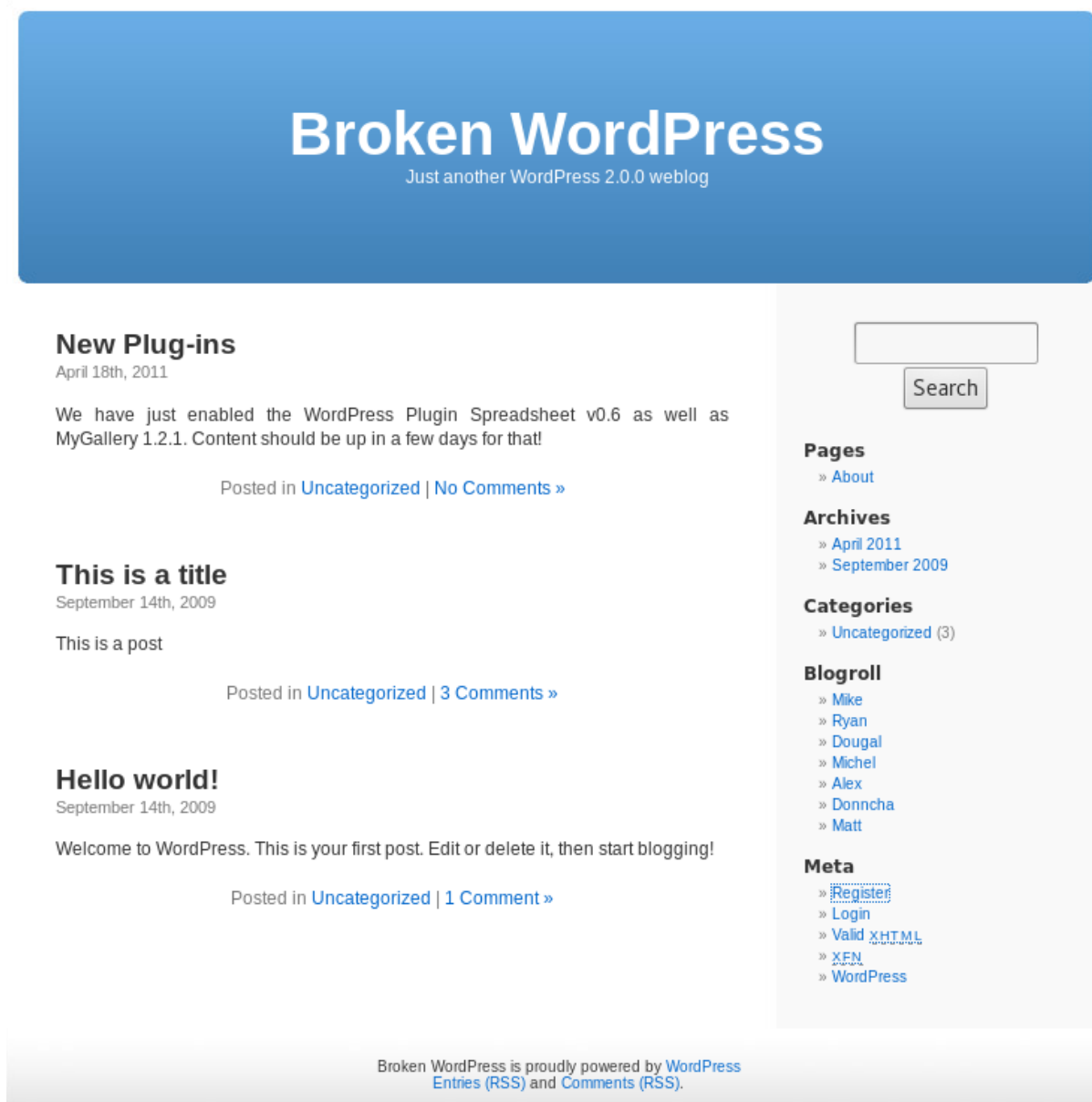


Figura 41 - Link para acesso a página de registro

A figura 42 mostra a tela de registro que apresenta dois campos para entrada de texto, *Username* e *E-mail*. O objetivo desse teste é verificar se esses campos estão vulneráveis a *cross-site scripting* refletido e mostrar uma exploração básica do problema.



The image shows a screenshot of the WordPress registration page. At the top, there is the WordPress logo (a 'W' in a circle) followed by the text 'WORDPRESS' in a bold, serif font. Below this, the text 'Register for this Blog' is displayed, with 'Blog' on a separate line. There are two input fields: one for 'Username:' and one for 'E-mail:'. Below the 'E-mail:' field, a message states 'A password will be emailed to you.' At the bottom right, there is a button labeled 'Register »'. At the bottom left, there are three links: '< Back to blog', 'Login', and 'Lost your password?'.

Figura 42 - Página de registro para teste

Após acessar a tela de registro é preciso clicar no botão *Register* para disparar uma requisição HTTP do tipo POST ao servidor da aplicação, porém ela será interceptada pelo proxy da ferramenta Owasp Zap, permitindo uma análise dos parâmetros da requisição.

Conforme a figura 43, ao clicar na requisição que foi realizada e presente na aba de Histórico da Owasp Zap, é possível verificar que no corpo da requisição estarão presentes as entradas de dados *user_login* e *user_email*. Essa ferramenta será usada também para testar a vulnerabilidade dessas entradas, pois ela apresenta um conjunto de *strings* de ataque prontas para teste, facilitando o serviço manual de verificar a vulnerabilidade dos campos.

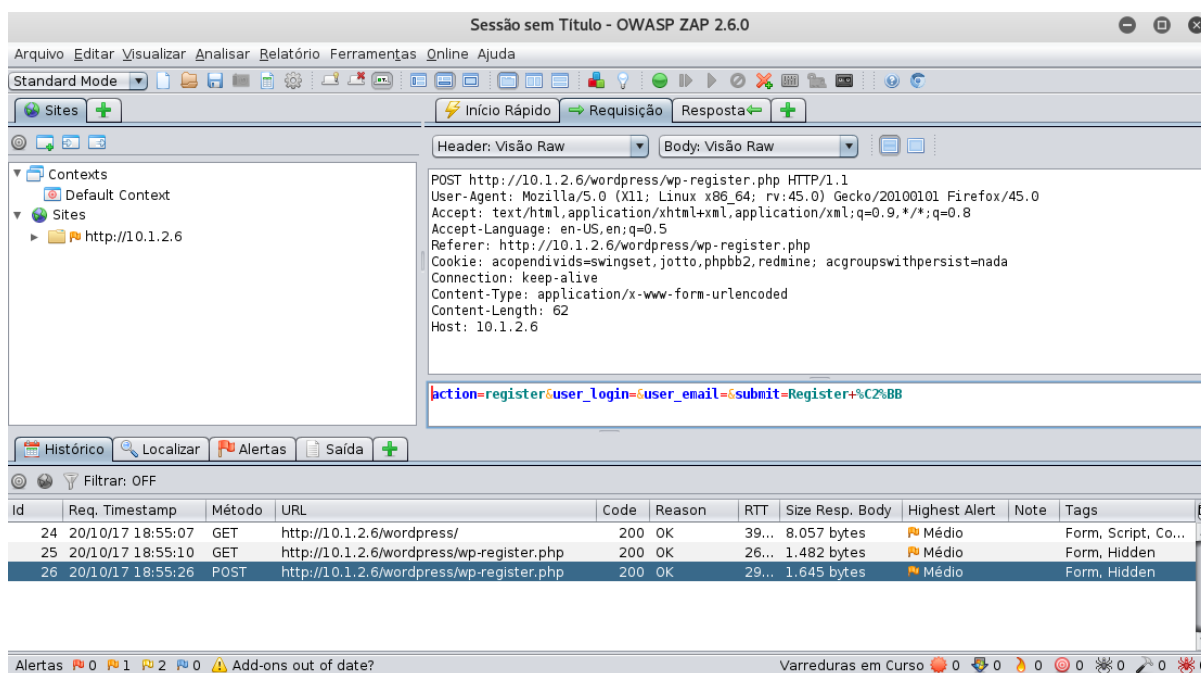


Figura 43 - Registro da requisição na Owasp Zap

Após selecionar a requisição basta clicar nela e na opção Ataque escolher o item *Active Scan*, assim como na figura 44. Essa opção irá realizar uma varredura por vulnerabilidades de vários tipos, disparando várias vezes a requisição e testando seus parâmetros. Na tela que abrir basta clicar em *Start Scan*.

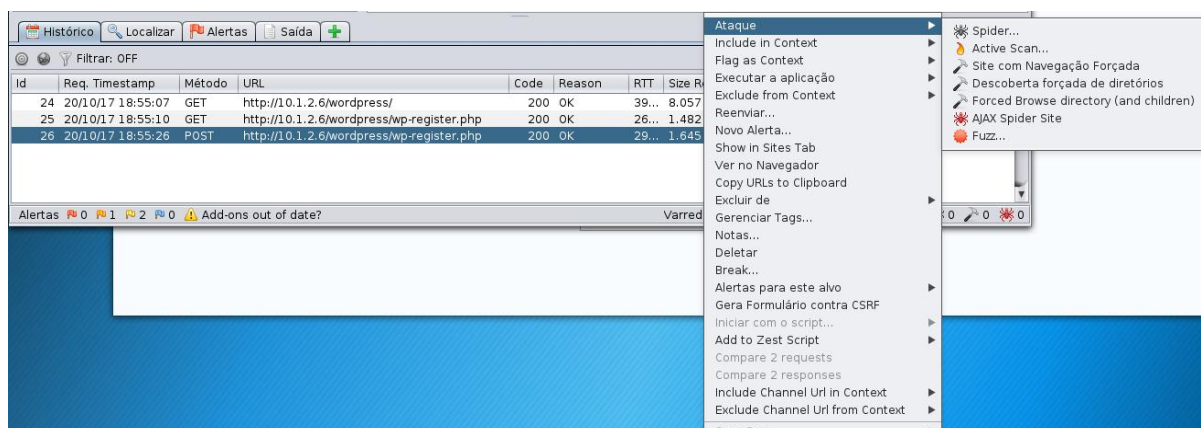


Figura 44 - Execução de varredura ativa

Quando a varredura ativa for concluída é possível ver na aba Alertas se alguma vulnerabilidade foi encontrada. O resultado dessa análise em cima da tela de registro da aplicação WordPress retorna como resultado duas vulnerabilidades de XSS refletido, uma para o campo de usuário e outra para o campo de e-mail. Na figura 45 pode-se visualizar as vulnerabilidades encontradas e quais dados de entrada foram utilizados no teste da ferramenta.

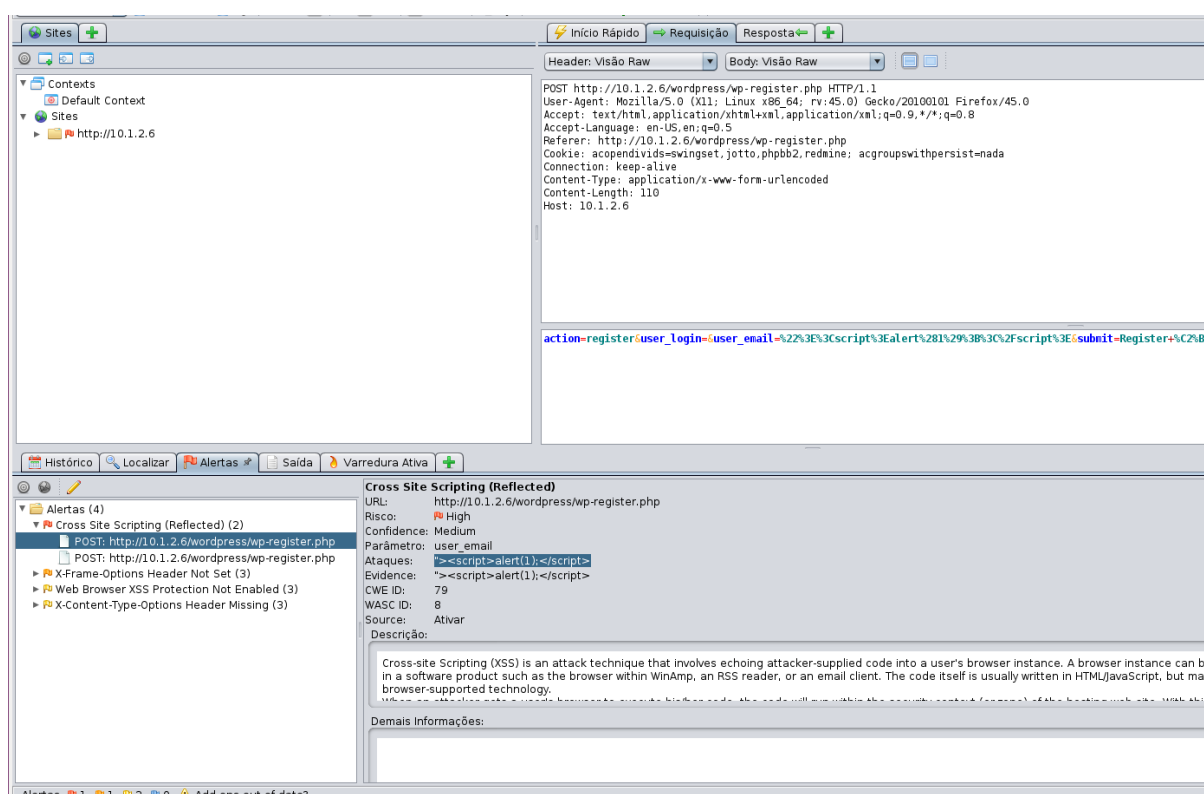


Figura 45 - Resultado da varredura ativa

A *string* de ataque utilizada pela ferramenta Owasp Zap foi `"><script>alert(1);</script>`, que mostra a falta de tratamento dos dados de e-mail e usuário enviados à aplicação, com isso é possível executar um *script* qualquer no navegador de um usuário que executar uma requisição contendo dados maliciosos.

Na tela da figura 46 uma caixa de alerta é exibida ao tentar registrar como *E-mail* o valor "><script>alert(`Cross-site scripting refletido`);</script>".

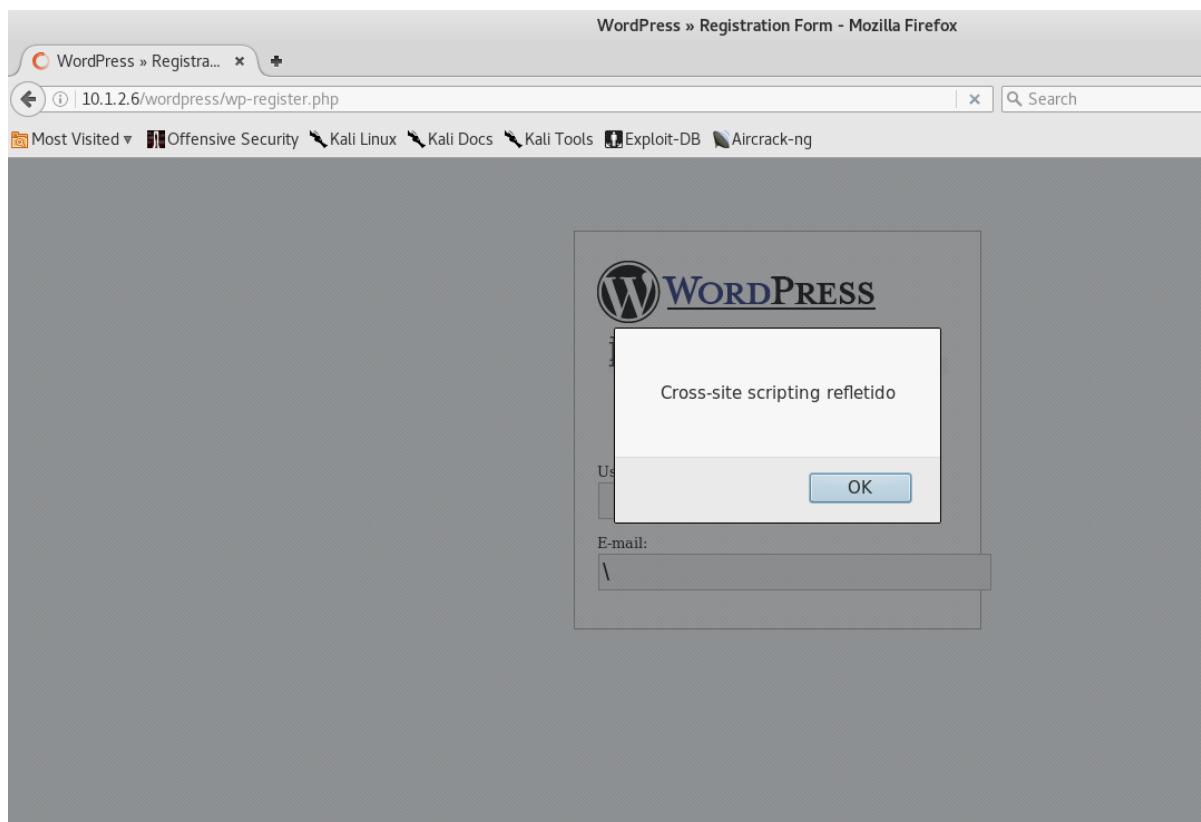


Figura 46 - Exploração de vulnerabilidade de XSS Refletido

4.3.2 Teste de *Cross-Site Scripting* Persistido

O processo de ataque de *cross-site scripting* persistido consiste geralmente das seguintes etapas (OWASP, 2014):

- Ao detectar uma página vulnerável, o atacante salva um código malicioso na aplicação, que possivelmente será armazenado num banco de dados;
- Um usuário realiza autenticação e visita uma página que recupera os dados maliciosos salvos na aplicação;
- Ao carregar a página, o código salvo pelo atacante será executado no navegador da vítima;
- O código pode dar acesso aos recursos da máquina do usuário, baixar arquivos, permitir roubo de informações sensíveis.

Testar se uma aplicação é vulnerável a esse tipo de ataque é bem semelhante ao processo de teste de XSS refletido. Pode-se descrever o teste de XSS persistido em algumas fases.

A primeira fase é explorar a aplicação em busca dos pontos de entrada de dados que serão salvos. Geralmente páginas que apresentam formulários de cadastro e edição de usuários, fóruns de mensagens e blogs que permitem usuários deixar comentários são bons exemplos de entrada de dados que devem ser investigadas.

A segunda fase trata da análise do código da página. Dados armazenados numa aplicação geralmente podem ser visualizados na própria aplicação e inseridos em *tags* HTML ou utilizados em *scripts* nas próprias páginas. A análise do código permite entender se campos presentes na página são armazenados pela aplicação e quais dados são inseridos na página e onde são.

O local onde um dado salvo é disponibilizado numa tela é importante pois é onde o código malicioso será inserido. Nessa etapa também é importante identificar quais requisições e seus parâmetros acontecem quando tenta-se salvar dados.

A terceira fase consiste em testar e avaliar o resultado dos testes em cima dos campos encontrados na fase anterior e que são armazenados na aplicação. O intuito é verificar se a entrada de dados não é filtrada, se é possível salvar um código malicioso na aplicação e carregá-lo em alguma página.

Execução do teste na aplicação OWASP Mutillidae II

Para realizar o teste de *cross-site scripting* persistido foi escolhida a aplicação Mutillidae II, encontrada na Owasp Broken. Essa aplicação fornece um ambiente onde é possível treinar e explorar diversos tipos de vulnerabilidades web.

Para o teste deve-se abrir a aplicação Mutillidae através do navegador na máquina Kali Linux no endereço 10.1.2.6. Lembrar que conforme a configuração do ambiente é necessário que a aplicação Owasp Zap esteja aberta, pois ela irá servir de *proxy* entre a aplicação e o navegador.

A Mutillidae apresenta um menu lateral contendo inúmeros exemplos vulneráveis para prática de teste de intrusão. Para realizar o teste de XSS persistido é preciso abrir a opção de menu conforme a figura 47.

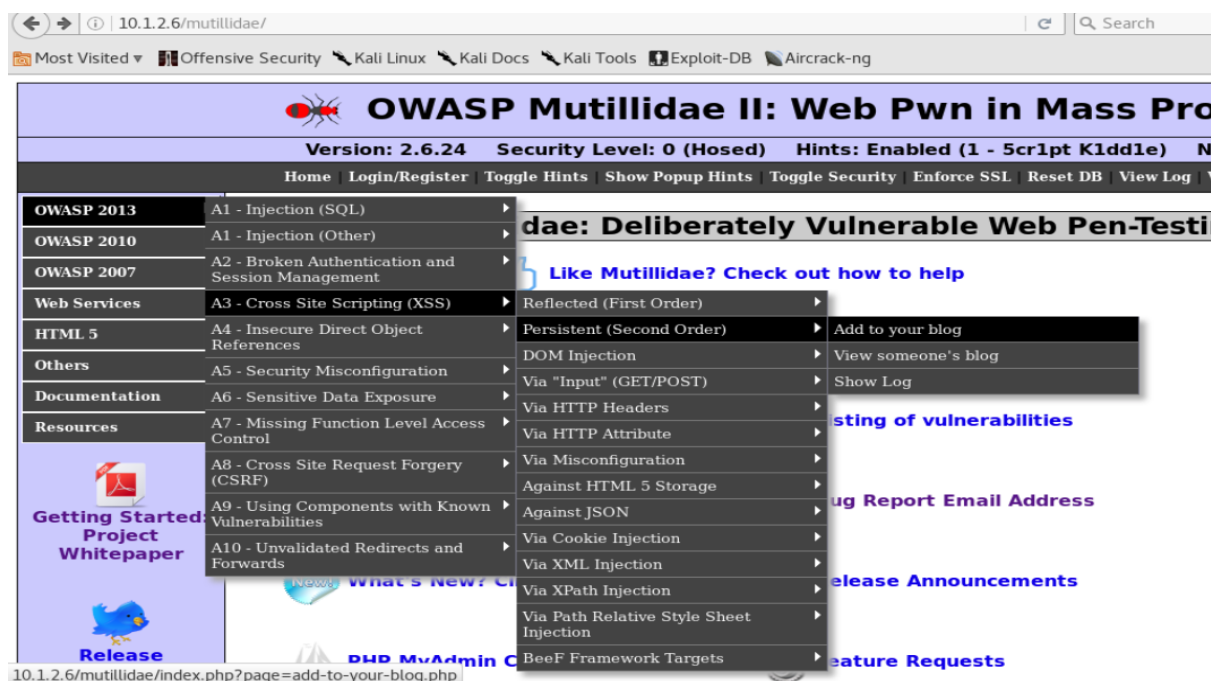


Figura 47 - Blog para teste na Mutillidae

A tela que abrir na Mutillidae apresenta um campo para salvar um comentário num blog e uma tabela de comentários salvos, conforme a figura 48.

Add New Blog Entry

[View Blogs](#)

Add blog for anonymous

Note: , <i> and <u> are now allowed in blog entries

View Blogs

1 Current Blog Entries			
	Name	Date	Comment
1	anonymous	2009-03-01 22:27:11	An anonymous blog? Huh?

Figura 48 - Campo de comentário em blog

Para testar a aplicação basta digitar no campo de comentário a frase “Este é um comentário de teste.” e clicar em *Save Blog Entry*. Após salvar o comentário é

possível ver que a tabela foi atualizada com o novo dado e isso levanta a suspeita de que o dado enviado ao servidor da aplicação para ser salvo é carregado na página.

Na Owasp Zap selecionar a requisição que foi realizada para salvar o comentário, figura 49, e executar uma varredura ativa em busca de vulnerabilidades nas entradas de dados.

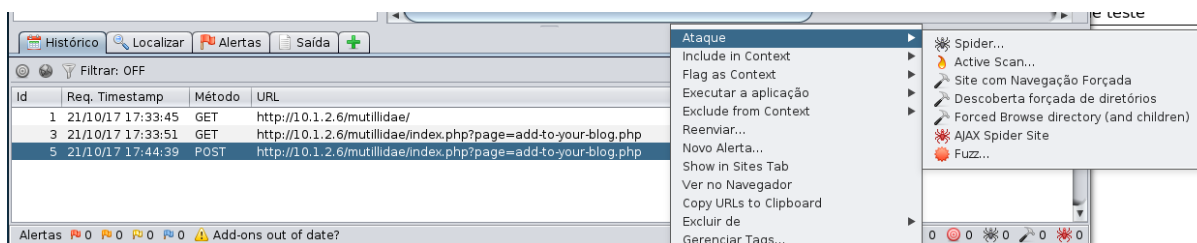


Figura 49 - Varredura ativa ao salvar comentário

Na aba de alertas, mostrada na figura 50, estará registrada uma ocorrência de *cross-site scripting* persistido, informando que o parâmetro `blog_entry` está vulnerável. Esse parâmetro contém o comentário a ser salvo.

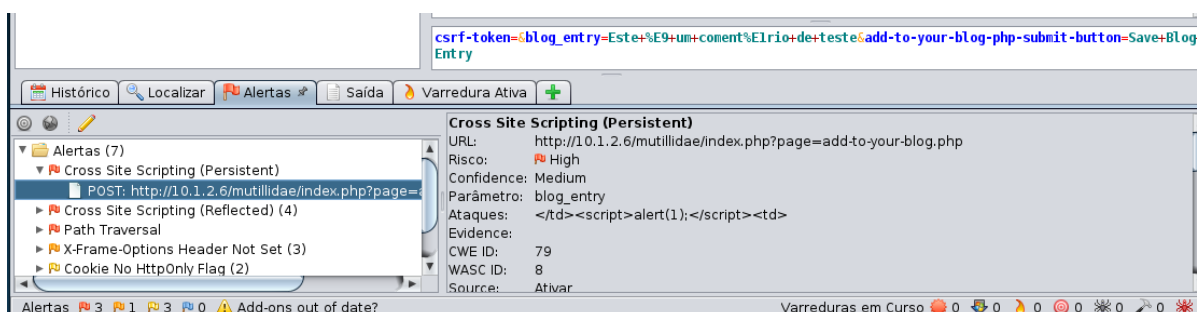


Figura 50 - Campo vulnerável a XSS persistido

Ao saber da vulnerabilidade deve-se inspecionar a página no navegador para saber onde exatamente o dado salvo está sendo carregado, com o intuito de formular uma *string* de ataque funcional. Através da análise do código HTML é possível verificar que o comentário é carregado na *tag* HTML presente na figura 51.

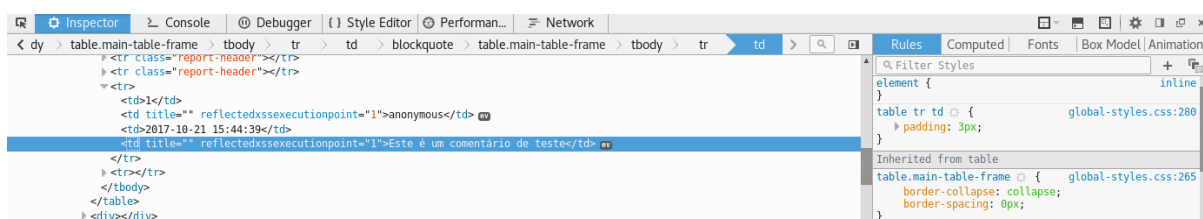



Figura 51 - Código HTML da página de comentários do blog

Com a análise do código percebe-se que é possível inserir um dado malicioso no formato `</td><script>alert(`Cross-site scripting persistido`);</script><td>` no campo de comentário e salvar.

Antes de tentar realizar o ataque é necessário retornar a página inicial da Mutillidae e limpar o banco de dados da aplicação através da opção *Reset DB*, no menu superior, após isso aparecerá uma caixa de confirmação, basta clicar em *OK*. Essa limpeza deve ser feita, pois a ferramenta Owasp Zap ao testar a requisição que salva comentários no blog registrou vários comentários, incluindo comandos maliciosos.

Para concluir o teste, na tela de comentários do blog basta inserir a *string* de ataque formulada anteriormente no campo de comentário e salvar, conforme figura 52.

Na figura 53, como resultado do teste, a aplicação salva um código malicioso que será carregado no navegador do usuário que acessar a página vulnerável e exibirá um alerta com a mensagem “Cross-site scripting persistido”.



Add blog for anonymous

Note: , <i> and <u> are now allowed in blog entries

`</td><script>alert('Cross-site scripting persistido');</script><td>`

Save Blog Entry

Figura 52 - Salvar código malicioso como comentário

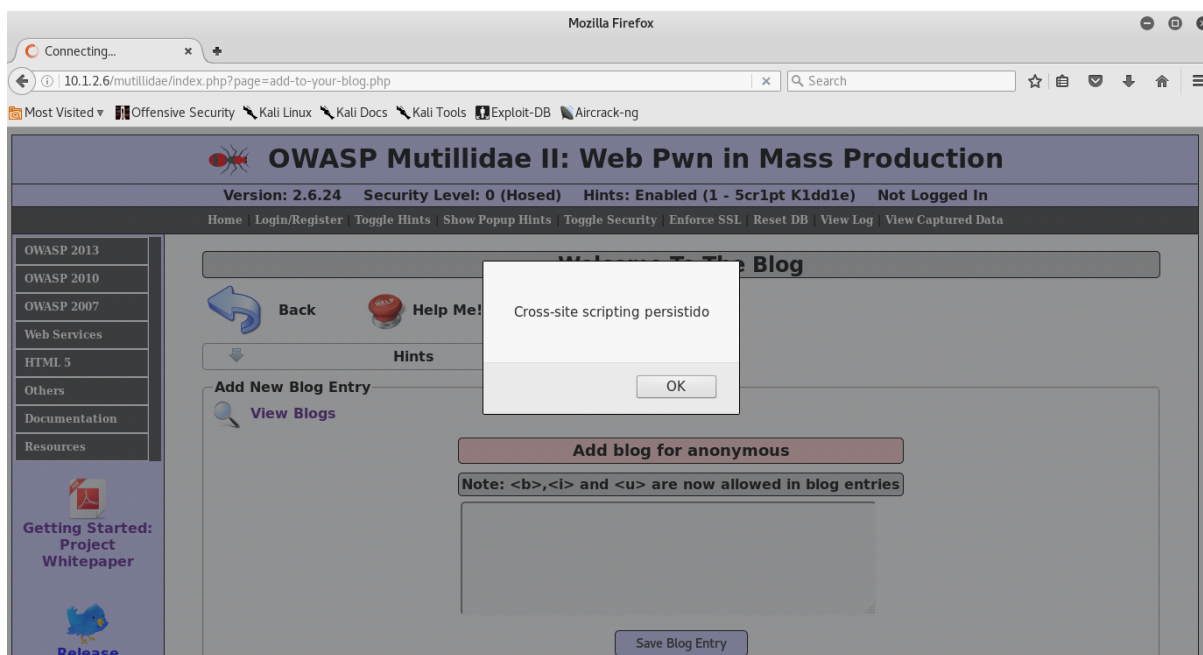


Figura 53 - Exploração de vulnerabilidade de XSS Persistido

4.3.3 Teste de SQL *Injection*

Segundo a OWASP (2014), para realizar um ataque de SQL *Injection* um atacante, sabendo de entrada de dados que provavelmente são utilizados como parte de uma *query* SQL a ser executada pela aplicação, realiza os seguintes passos:

- Constrói o dado de forma a gerar uma *query* SQL correta e que realize o que ele quer na aplicação. Inserindo esse dado em campos vulneráveis ou parâmetros da requisição;
- Se a aplicação retornar uma mensagem de erro informando que o comando SQL executado está incorreto, torna-se mais fácil para o atacante reconstruir o comando.
- Se a aplicação mascarar o erro, o atacante tentará descobrir a lógica utilizada pelo desenvolvedor da aplicação ao construir o comando SQL.
- Realiza a injeção pela execução da requisição vulnerável da aplicação, obtendo dados ou privilégio de acesso.

Um ataque de injeção SQL pode ainda ser dividido em três classes (OWASP, 2014):

- *Inband*: A injeção do código SQL malicioso acontece na mesma página em que o resultado da execução do comando é apresentado.
- *Out-of-band*: O resultado da query executada com a injeção é enviado através de outro canal, como e-mail.

- Inferencial ou *Blind*: O atacante não recupera dados do banco de dados, porém é capaz de inferir informações através da análise dos resultados das requisições realizadas ao servidor do banco de dados.

Para verificar se uma aplicação é vulnerável a injeção de SQL, basicamente devem ser realizados três passos. Na primeira etapa busca-se entender a interação entre a aplicação testada e seu banco de dados, verificando quais as formas que a aplicação solicita acesso e salva dados, seja por meio de requisições ou campos de dados em páginas. Formulários de autenticação, de cadastro e campos de busca são ótimos exemplos de canais que levam a aplicação executar consultas numa base de dados.

Na segunda etapa são utilizados dados construídos para fazer com que a aplicação retorne erros do banco de dados na própria página. Esses dados são usados diretamente nos campos ou como parâmetros das requisições a serem testadas. Como exemplo é possível citar a inserção do caractere ; ou ' como dados, que se concatenados no comando SQL presente na aplicação, tornam esse comando inválido e ao ser executado um erro é gerado e retornado.

Ainda na segunda etapa, usando caracteres especiais como entrada de dados é possível verificar se a aplicação realiza filtros adequados nos campos e caso ela não filtre os dados e um erro seja retornado, a vulnerabilidade já está comprovada.

Os erros gerados por comandos SQL inválidos e que são mostrados na tela, geralmente mostram informações de qual o tipo de banco de dados retornou um erro e em alguns casos a especificação do erro de sintaxe. Existem aplicações que

exibem esses erros de maneira diferente, como o famoso erro interno do servidor 500.

No último passo, ao saber da vulnerabilidade em determinado campo, deve-se construir a *string* de ataque e executar a injeção SQL para validar se a vulnerabilidade traz algum risco ou não à aplicação.

Execução do teste na aplicação Mutillidae II

Para o teste deve-se abrir a aplicação Mutillidae através do navegador na máquina Kali Linux no endereço 10.1.2.6. A página que será testada está localizada no caminho da figura 54. Essa página apresenta um formulário que dado um *name* e *password* retorna informações a respeito de um usuário salvo na base de dados da aplicação.

OWASP Mutillidae II: Web Pwn in Mass Pro

Version: 2.6.24 Security Level: 0 (Hosed) Hints: Enabled (1 - 5cr1pt K1dd1e) N

Home Login/Register Toggle Hints Show Popup Hints Toggle Security Enforce SSL Reset DB View Log

OWASP 2013	A1 - Injection (SQL)	SQLi - Extract Data	User Info (SQL)
OWASP 2010	A1 - Injection (Other)	SQLi - Bypass Authentication	
OWASP 2007	A2 - Broken Authentication and Session Management	SQLi - Insert Injection	
Web Services	A3 - Cross Site Scripting (XSS)	Blind SQL via Timing	
HTML 5	A4 - Insecure Direct Object References	SQLMAP Practice	
Others	A5 - Security Misconfiguration	Via JavaScript Object Notation (JSON)	
Documentation	A6 - Sensitive Data Exposure	Via SOAP Web Service	
Resources	A7 - Missing Function Level Access Control	Via REST Web Service	
	A8 - Cross Site Request Forgery (CSRF)		
	A9 - Using Components with Known Vulnerabilities		
	A10 - Unvalidated Redirects and Forwards		

User Lookup (SQL)

Please enter username and password to view account details

Name

Password

[View Account Details](#)

Don't have an account? [Please register here](#)

Figura 54 - Formulário de busca de dados dos usuários na Mutillidae

Examinando a tela é possível ver que existem dois campos necessários para realizar a busca de um usuário. Como se trata de uma busca, um atacante pode inferir que provavelmente os dados serão carregados de uma base de dados, portanto ele testará se os campos da página filtram caracteres especiais. O atacante também testará como os dados informados são usados na requisição à aplicação, para tentar descobrir como são usados na consulta SQL.

Assim como na figura 55, ao digitar o dado teste no campo *Name* e *Password* e clicar em *View Account Details*, uma requisição HTTP é disparada e os dados informados são colocados como parâmetros na URL. O resultado da requisição à aplicação foram 0 registros para o *name* teste.

The screenshot shows the OWASP Mutillidae II web application interface. The browser's address bar displays the URL: `10.1.2.6/mutillidae/index.php?page=user-info.php&username=teste&password=teste&user-info-php-submit-button=View+Account+Details`. The application header includes the title "OWASP Mutillidae II: Web Pwn in Mass Production" and version information: "Version: 2.6.24 Security Level: 0 (Hosed) Hints: Enabled (1 - 5cr1pt K1dd1e) Not Lo". A navigation bar contains links: Home, Login/Register, Toggle Hints, Show Popup Hints, Toggle Security, Enforce SSL, Reset DB, View Log, and View C.

The main content area is titled "User Lookup (SQL)". It features a "Back" button with a blue arrow icon and a "Help Me!" button with a red button icon. Below these is a "Hints" section with a dropdown arrow. There are two links: "Switch to SOAP Web Service version" with an AJAX icon and "Switch to XPath version" with an XML icon.

A red dashed box contains the message: "Authentication Error: Bad user name or password". Below this is a pink box with the text: "Please enter username and password to view account details". The form has two input fields: "Name" with the value "teste" and "Password" with masked characters "*****". A "View Account Details" button is located below the password field.

At the bottom, a link says: "Dont have an account? Please register here". The footer displays the result: "Results for 'teste', 0 records found."

Figura 55 - Busca de usuário teste

Na ferramenta Owasp Zap ao fazer uma varredura ativa na URL da requisição de busca do usuário, é possível ver na aba de alertas que foram encontradas duas vulnerabilidades de injeção de SQL, uma em cada campo da tela, figura 56.

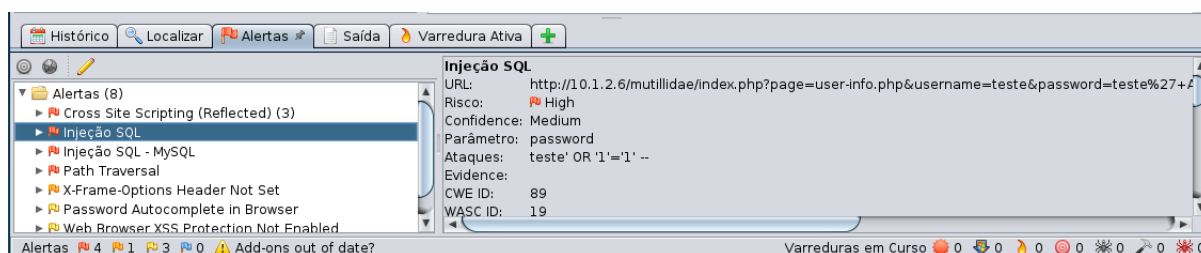


Figura 56 - Alertas de injeção SQL

Sabendo que os campos estão vulneráveis é possível tentar inserir o caractere ' no campo de *Name*, ver imagem 57, para quebrar a consulta SQL e verificar se é retornando algum erro do banco de dados na tela. Na figura 58 é exibido o erro gerado ao tentar pesquisar o usuário.

**Please enter username and password
to view account details**

Name

Password

Dont have an account? [Please register here](#)

Figura 57 - Formulário do teste de injeção SQL

Please enter username and password to view account details

Name

Password

[View Account Details](#)

[Dont have an account? Please register here](#)

Error Message

Failure is always an option

Line	170
Code	0
File	/owaspbwa/utillidae-git/classes/MySQLHandler.php
Message	<p>/owaspbwa/utillidae-git/classes/MySQLHandler.php on line 165: Error executing query:</p> <p>connect_errno: 0</p> <p>errno: 1064</p> <p>error: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '''' AND password='' at line 2</p> <p>client_info: 5.1.73</p> <p>host_info: Localhost via UNIX socket</p> <p>) Query: SELECT * FROM accounts WHERE username='' AND password='' (0) [Exception]</p>
Trace	<p>#0 /owaspbwa/utillidae-git/classes/MySQLHandler.php(283): MySQLHandler->doExecuteQuery('SELECT * FROM a...') #1 /owaspbwa/utillidae-git/classes/SQLQueryHandler.php(327): MySQLHandler->executeQuery('SELECT * FROM a...') #2 /owaspbwa/utillidae-git/user-info.php(191): SQLQueryHandler->getUserAccount('', '') #3 /owaspbwa/utillidae-git/index.php(614): require_once('/owaspbwa/utill...') #4 {main}</p>
Diagnostic Information	Error attempting to display user information

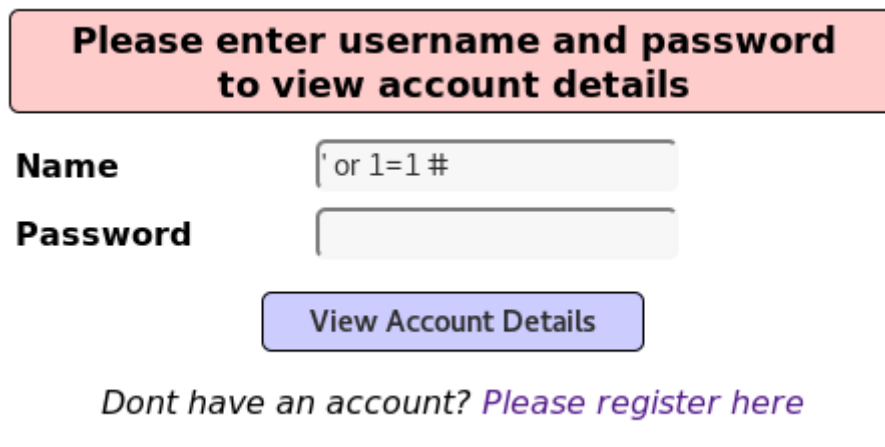
[Click here to reset the DB](#)

Figura 58 - Erro do banco de dados

Verificando a mensagem do erro é possível ver que o comando SQL executado pela aplicação é o seguinte:

`SELECT * FROM accounts WHERE username='' AND password=''`

Conforme a figura 59, para esse teste basta digitar como valor do campo *Name* a *string* de ataque ' or 1=1 #.



Please enter username and password to view account details

Name

Password

[View Account Details](#)

Dont have an account? [Please register here](#)

Figura 59 - *String* de ataque de injeção SQL

Ao tentar buscar um usuário com esse *name* a aplicação irá inserir o dado no comando SQL e ele deverá ficar da seguinte maneira:

```
SELECT * FROM accounts WHERE username=" or 1=1 #" AND password="
```

A *string* de ataque formulada aplicou uma condição booleana, informando para o banco de dados retornar todos os usuários cujo *username* seja igual a vazio ou então caso 1 seja igual a 1, tornando a expressão sempre verdadeira. Como resultado da consulta todos os dados de todos os usuários são retornados, ver figura 60. Gravíssima falha de segurança. O caractere # para o sistema gerenciador de banco de dados MySql trata-se de um comentário, portanto o restante do comando é ignorado.

Results for "' or 1=1 #".25 records found.

```

Username=admin
Password=admin
Signature=g0t r00t?

Username=adrian
Password=somepassword
Signature=Zombie Films Rock!

Username=john
Password=monkey
Signature=I like the smell of confunk

Username=jeremy
Password=password
Signature=d1373 1337 speak

Username=bryce
Password=password
Signature=I Love SANS

Username=samurai
Password=samurai
Signature=Carving fools

Username=jim
Password=password
Signature=Rome is burning

Username=bobby
Password=password
Signature=Hank is my dad

Username=simba
Password=password
Signature=I am a super-cat

Username=dreveil
Password=password
Signature=Preparation H

```

Figura 60 - Resultado da injeção de código SQL

4.4 EXECUÇÃO DOS TESTES DE AUTENTICAÇÃO

4.4.1 Teste para burlar esquema de autenticação com injeção SQL

Um sistema de autenticação implementando de maneira incorreta e que não respeite medidas de segurança adequadas pode ser facilmente enganado, dando acesso a funcionalidades da aplicação à usuários não autorizados.

Segundo a OWASP (2014), um atacante pode ultrapassar um esquema de autenticação através da adulteração de requisições, modificando parâmetros na

URL utilizados para verificar um *login* com sucesso, construindo dados a serem usados nos campos de um formulário e roubando ou falsificando ids de sessão.

A execução desse teste consiste nas seguintes etapas:

- Verificar o formulário de *login* de uma aplicação, em busca de vulnerabilidades de injeção SQL;
- Após descobrir a vulnerabilidade, deve-se construir uma *string* de ataque SQL que realize a autenticação de um usuário qualquer ou sem nem informar usuário;
- Realizar o *login* no sistema com o código malicioso.

Execução do teste na aplicação WackoPicko

WackoPicko é uma aplicação desenvolvida para testes de segurança, simulando uma aplicação real de um sistema de *upload* de imagens, contendo autenticação de usuários e comentários. Na figura 61 pode-se ver a tela inicial da aplicação, presente na OWASP Broken.

Home Upload Recent Guestbook Login

Search

Welcome to WackoPicko

On WackoPicko, you can share all your crazy pics with your friends. But that's not all, you can also buy the rights to the high quality version of someone's pictures. WackoPicko is fun for the whole family.

New Here?

[Create an account](#)

[Check out a sample user!](#)

[What is going on today?](#)

Or you can test to see if WackoPicko can handle a file:

Check this file: No file selected.

With this name:

[Home](#) | [Admin](#) | [Contact](#) | [Terms of Service](#)

Figura 61 - Tela inicial da aplicação WackoPicko

Ao tentar acessar a tela de *upload* ou abrir alguma imagem na tela de fotos recentes, a aplicação será redirecionada para tela de autenticação, impedindo o acesso de usuário não autenticado.

Após clicar na opção *Login* presente no menu, um formulário para autenticação deve abrir, conforme a figura 62. Clique no botão de *login* para realizar uma requisição à aplicação.

Na OWASP Zap faça uma varredura ativa na requisição que acabou de ser disparada, para tentar encontrar uma vulnerabilidade de injeção SQL nos campos do formulário da WackoPicko.

Figura 62 - Formulário de *login* WackoPicko

Como resultado da varredura foram detectadas algumas vulnerabilidades. Na aba de alertas é possível ver injeção de SQL no campo *Username*, figura 63.

Injeção SQL

URL: http://10.1.2.6/WackoPicko/users/login.php

Risco: High

Confidence: Medium

Parâmetro: username

Ataques: ' OR '1'='1' --

Evidence:

CWE ID: 89

WASC ID: 19

Source: Ativar

Descrição:

SQL injection may be possible.

Demais Informações:

The page results were successfully manipulated using the boolean conditions [' AND '1'='1' --] and [' OR '1'='1' --]
The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison
Data was NOT returned for the original parameter.

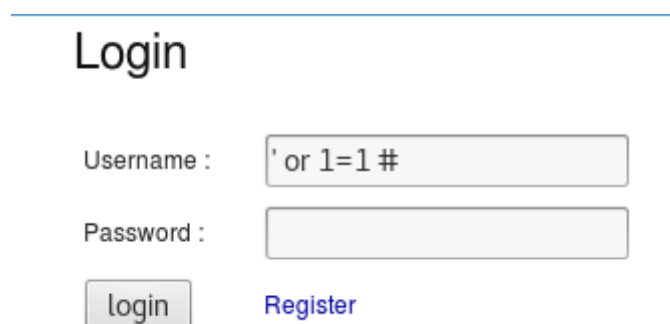
Figura 63 - Resultado de varredura na WackoPicko

Na tela do formulário ao tentar inserir o caractere ' e realizar o *login*, um erro do banco de dados é retornado na tela, assim como na figura 64, na qual podemos ver que se trata de um banco de dados Mysql e o erro de sintaxe SQL.

You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near "" and `password` = SHA1(CONCAT("", `salt`)) limit 1' at line 1

Figura 64 - Erro na consulta SQL de *login* WackoPicko

Novamente na tela do formulário, uma *string* de ataque é criada e inserida no campo de *Username*. O dado ' or 1=1 # pode ser inserido como valor do campo. Ver figura 65.



Login

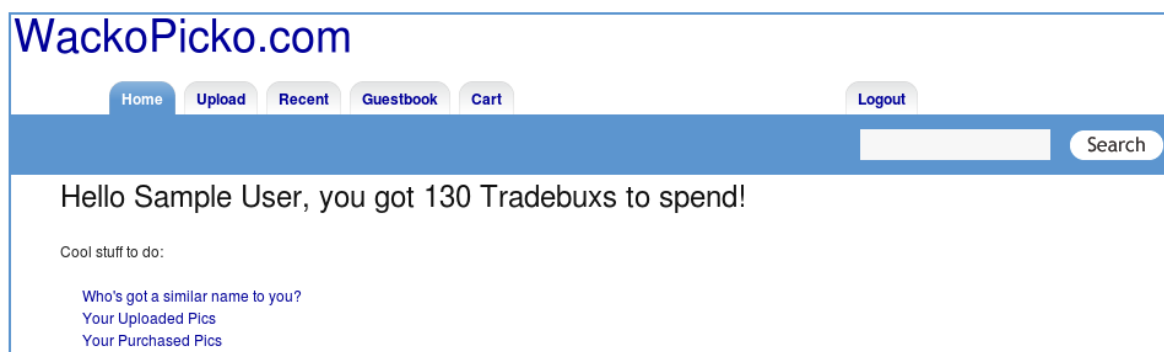
Username :

Password :

[Register](#)

Figura 65 - Tentativa de *login* na aplicação WackoPicko

Ao realizar a requisição de *login*, o esquema de autenticação deve ter sido burlado e um determinado usuário retornado pela consulta SQL da aplicação estará autenticado no sistema, conforme figura 66.



WackoPicko.com

Home Upload Recent Guestbook Cart Logout

Search

Hello Sample User, you got 130 Tradebuxs to spend!

Cool stuff to do:

- [Who's got a similar name to you?](#)
- [Your Uploaded Pics](#)
- [Your Purchased Pics](#)

Figura 66 - Esquema de autenticação burlado por injeção SQL

4.4.2 Teste para burlar esquema de autenticação via *cookie*

Algumas aplicações mantêm dados armazenados nos *cookies* do navegador. Dados como identificador de sessão podem ser utilizados posteriormente para verificar se um usuário está autenticado com uma sessão válida.

Neste teste será possível alterar a sessão de modo que com um usuário logado se tenha acesso a outro usuário, sem precisar fazer autenticação, simplesmente trocando dados armazenados nos *cookies*.

Os passos a serem executados são:

- Realizar um registro para autenticação válida numa aplicação;
- Se autenticar na aplicação para analisar os *cookies* salvos por ela;
- Alterar alguma informação de forma a burlar o sistema de autenticação.

Execução do teste na aplicação Mutillidae II

Para executar o teste deve-se abrir a página do caminho especificado na figura 67 na Mutillidae. Trata-se da página de *login* da aplicação. Depois de abrir a página é necessário registrar um usuário na aplicação, clicando em *Please register here* conforme figura 68.



Figura 67 - Caminho para acessar página de teste

Please sign-in

Username

Password

Dont have an account? [Please register here](#)

Figura 68 - Registrar usuário na Mutillidae

Na tela de cadastro é necessário preencher os campos *Username*, *Password* e *Confirm Password*. De exemplo o usuário que será criado receberá o *username* bobtester com a senha teste. Ver figura 69.

Please choose your username, password and signature

Username	<input style="width: 90%;" type="text" value="bobtester"/>	
Password	<input style="width: 90%;" type="password" value="•••••"/>	Password Generator
Confirm Password	<input style="width: 90%;" type="password" value="•••••"/>	
Signature	<div style="border: 1px solid #ccc; height: 40px; width: 100%;"></div>	

Create Account

Figura 69 - Campos de usuário preenchidos

Após criada a conta, deve-se acessar a tela de *login* novamente e se autenticar com os dados recém cadastrados. Assim como na figura 70, após a autenticação a aplicação deve ser redirecionada para a página principal da Mutillidae com o usuário bobtester autenticado.

OWASP Mutillidae II: Web Pwn in Mass Production

Version: 2.6.24 Security Level: 0 (Hosed) Hints: Enabled (1 - 5cr1pt K1dd1e) Logged In User: **bobtester** ()

Home
Logout
Toggle Hints
Show Popup Hints
Toggle Security
Enforce SSL
Reset DB
View Log
View Captured Data

Mutillidae: Deliberately Vulnerable Web Pen-Testing Application

[Like Mutillidae? Check out how to help](#)

| Do?
[Video Tutorials](#)

[Listing of vulnerabilities](#)

[Bug Report Email Address](#)

Figura 70 - Autenticação do usuário bobtester

Com o usuário autenticado é necessário investigar os *cookies* do navegador Firefox, com o intuito de encontrar algum dado mantido pela aplicação. Para isso basta inspecionar a página e na aba *network* clicar em *Reload*, conforme figura 71.



Figura 71 - Inspeção da página da aplicação Mutillidae

Na aba *network* agora devem ter aparecido todas as requisições realizadas à aplicação para poder carregar a página. Ao clicar na primeira requisição é possível ver um novo *menu* com a aba *Cookies*. Na figura 72 é possível visualizar os *cookies*.

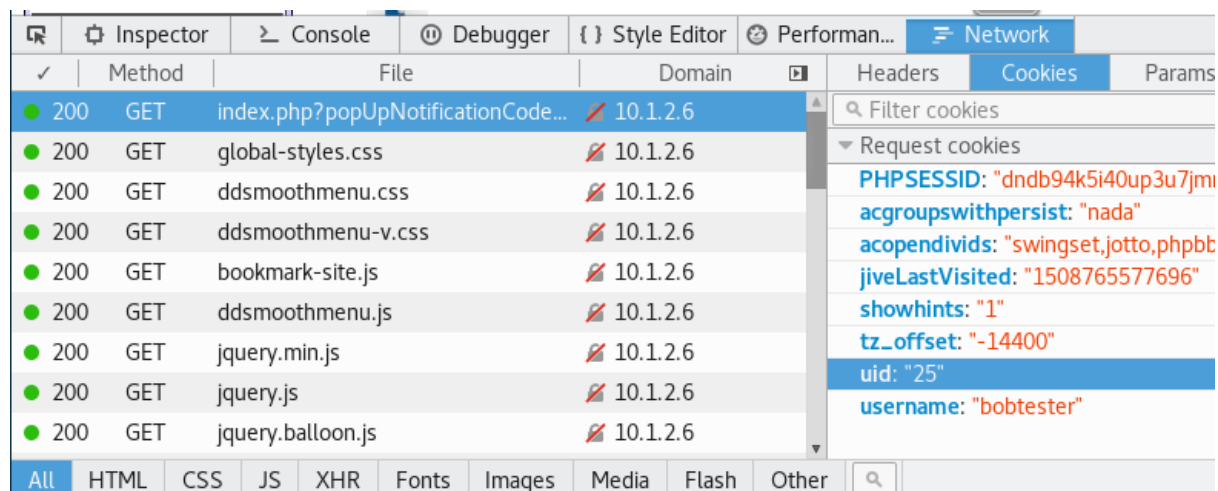


Figura 72 - Cookies da aplicação Mutillidae

Nos dados armazenados nos *cookies* do navegador é possível ver a chave *uid* com valor 25 para o *username* bobtester. Pode-se desconfiar que esse dado seja um identificador do usuário e para ter certeza é necessário criar uma nova conta. Para o novo usuário será cadastrado o *username* mariatester com a senha teste. É necessário fazer *logout* da aplicação para registrar um novo usuário.

Depois de realizado o cadastro e se autenticar com mariatester, a página deve ser inspecionada novamente para visualizar os *cookies* do navegador. Na figura 73 é possível ver que há um novo *uid* para o usuário autenticado e seu valor é de 26.

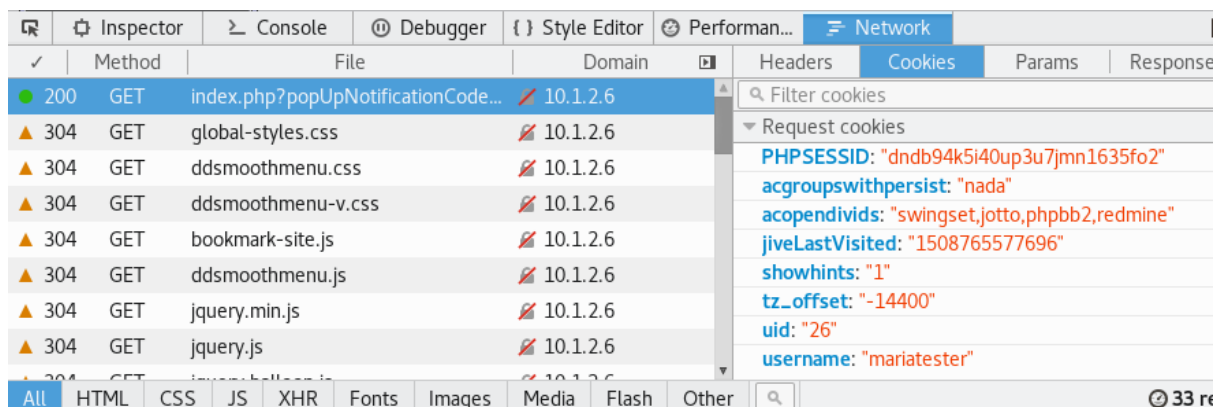


Figura 73 - Usuário mariatester autenticado na Mutillidae

Como o valor de uid aumentou uma unidade pode-se inferir que para cada cadastro novo esse identificador é incrementando uma unidade e que possivelmente existam usuários no banco de dados com identificadores inferiores à 26. Seguindo essa linha de raciocínio é possível tentar alterar esse valor para 10 nos *cookies* para verificar qual será o comportamento da aplicação.

Para alterar o valor de uid nos *cookies* é necessário executar na aba console o comando `document.cookie = 'uid=10'`, conforme a figura 74.

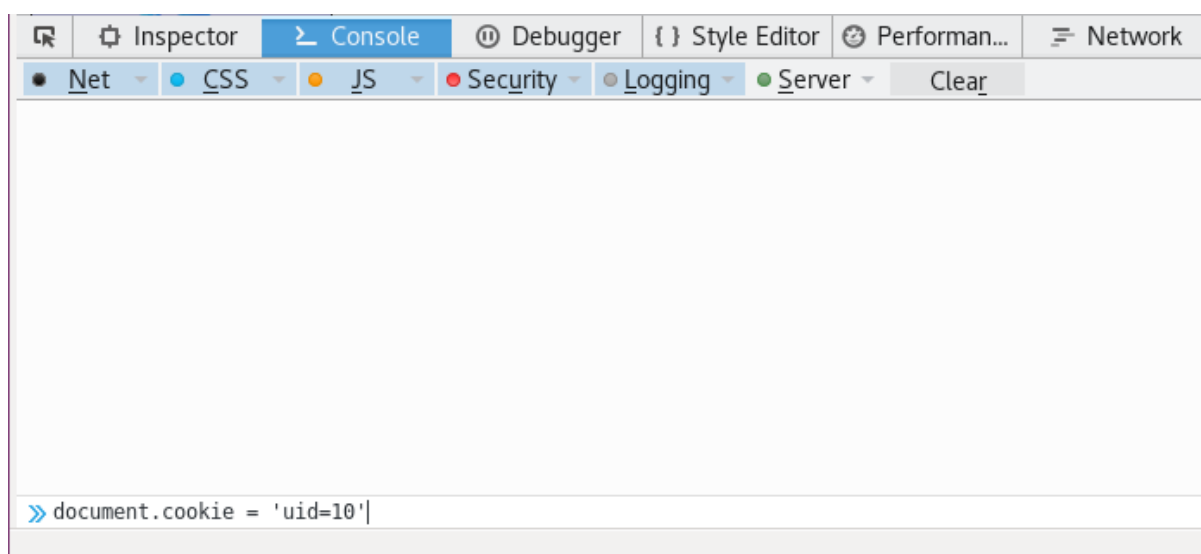


Figura 74 - Alteração do valor do *cookie*

Com o valor do *cookie* alterado deve ser realizada uma atualização na página para verificar qual o comportamento da aplicação. Na figura 75 pode-se perceber que ao atualizar a página o usuário autenticado foi alterado para dreveil.

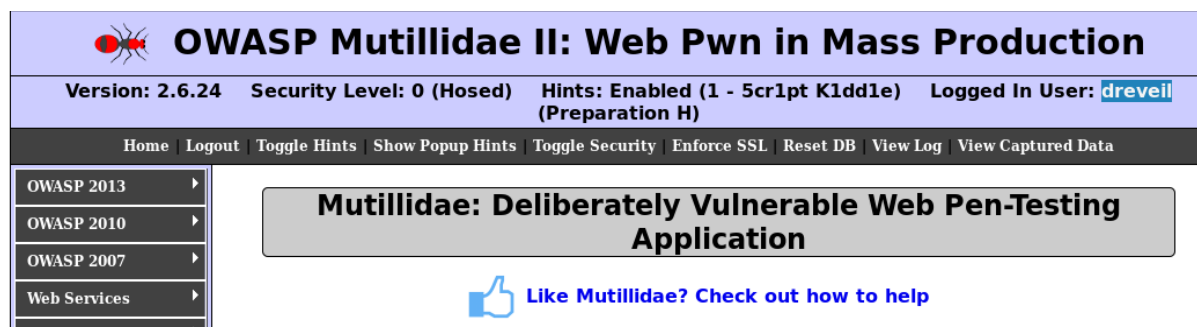


Figura 75 - Novo usuário autenticado na Mutillidae

Continuando na mesma linha dos identificadores de usuário, possivelmente os uids menores serão vinculados a contas de administradores da aplicação e com esse ataque será possível ter privilégios administrativos no sistema. Se o uid for alterado para 1, ao atualizar a página o usuário admin estará autenticado. Ver na figura 76.



Figura 76 - Usuário administrador autenticado

5 CONCLUSÃO

Neste trabalho foi desenvolvido um guia para realização de testes de segurança em aplicações web, destinado aos profissionais que trabalham com desenvolvimento de software, interessados pela área e alunos de graduação.

O guia aborda a detecção das vulnerabilidades de *cross-site scripting* persistido e refletido, de injeção de código e de autenticação, descritas no documento criado pela OWASP das 10 vulnerabilidades mais comuns em aplicações web. Para cobrir esses problemas de segurança foram realizados testes de autenticação e de validação de entrada de dados, selecionados a partir da metodologia da OWASP, em cima de aplicações desenvolvidas para testes de intrusão e sistemas reais em versões antigas e vulneráveis.

Com o auxílio deste guia, profissionais com pouco conhecimento em segurança de sistemas são capazes de testar suas aplicações e aprender como as vulnerabilidades abordadas surgem, como detectá-las através de ferramentas de varredura e como evitar.

Como trabalhos futuros é possível explorar os demais tipos de teste presentes na metodologia da OWASP complementando este guia, aplicar os testes em sistemas reais, elaborar guias para outras técnicas de testes como modelagem de ameaças e revisão de código fonte.

REFERÊNCIAS BIBLIOGRÁFICAS

AQUINO, M. C. Um resgate histórico do hipertexto: O desvio da escrita hipertextual provocado pelo advento da Web e o retorno aos preceitos iniciais através de novos suportes. **Unirevista**, [Porto Alegre], v. 1, n. 3, p.1-14, jul. 2006.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **ABNT NBR ISO/IEC 17799**: Tecnologia da informação — Técnicas de segurança — Código de prática para a gestão da segurança da informação. 2 ed. Rio de Janeiro: Abnt, 2005. 120 p.

BENNETTS, Simon. **OWASP Zed Attack Proxy Project**. Disponível em: <https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project>. Acesso em: 04 set. 2017.

BERNERS-LEE, T; MASINTER, L; MCCAILL, M. (Ed.). **RFC 1738**: Uniform Resource Locators. 1994. Disponível em: <<https://www.ietf.org/rfc/rfc1738.txt>>. Acesso em: 02 ago. 2017.

BISHOP, Matt. **Introduction to Computer Security**. Boston, Ma: Addison-wesley, 2005. 784 p.

BRAGA, A. M. Visão geral das boas práticas para construção de softwares seguros. **Revista Técnica IPEP**, São Paulo, SP, V. 7, N. 2, p. 65-78, jul./dez. 2007. ISSN 1807-8125.

CERN. **The World Wide Web project**. Disponível em: <<http://info.cern.ch/hypertext/WWW/TheProject.html>>. Acesso em: 02 ago. 2017.

CERT.br. **Cartilha de Segurança para Internet**: Versão 4.0. 2. ed. São Paulo: Comitê Gestor da Internet no Brasil, 2012. 123 p.

CERT.br. **Cartilha de Segurança para Internet**. Disponível em: <<https://cartilha.cert.br>>. Acesso em: 19 jun. 2017.

DI LUCCA, G. A; FASOLINO, A. R. Testing Web-based applications: The state of the art and future trends. **Information And Software Technology**, [S.l.], v. 48, n. 12, p.1172-1186, dez. 2006.

HELD, Gilbert (Ed.). **Server Management**. Boca Raton: Crc Press, 2000. 683 p.

KAPPEL, G. (2004). **Web Engineering - Old Wine in New Bottles? Invited Talk**. Fourth International Conference on Web Engineering (ICWE'04).

MENEZES, P. M; CARDOSO, L. M; ROCHA, F. G. Segurança em redes de computadores uma visão sobre o processo de Pentest. **Interfaces Científicas - Exatas e Tecnológicas**, [s.l.], v. 1, n. 2, p.85-96, 28 maio 2015.

MEUCCI, Matteo; MULLER, Andrew. **OWASP Testing Guide v4**. Maryland, Eua: Owasp, 2014. 222 p.

MOCELIM, E. F. **Manual de Auditoria baseado na Norma ICPEDU**. 2015. 40 f. TCC (Graduação) - Curso de Sistemas de Informação, Departamento de Informática e Estatística, Universidade Federal de Santa Catarina, Florianópolis, 2015.

OFFENSIVE SECURITY. **What is Kali Linux ?** Disponível em: <<https://docs.kali.org/introduction/what-is-kali-linux>>. Acesso em: 01 set. 2017.

OLIVEIRA, T. S. T. **Testes de segurança em aplicações web segundo a metodologia OWASP**. 2012. 126 f. TCC (Graduação) - Curso de Ciência da Computação, Universidade Federal de Lavras, Lavras, 2012.

OWASP. **Open Web Application Security Project**. Disponível em: <<https://www.owasp.org>>. Acesso em: 05 abr. 2017.

OWASP. **OWASP Top 10**: Os dez riscos de segurança mais críticos em aplicações web. São Paulo: Owasp, 2013. 23 p.

PEIXINHO, Ivo de Carvalho; FONSECA, Francisco Marmo da; LIMA, Francisco Marcelo Marques. **Segurança de Redes e Sistemas**. Rio de Janeiro: RNP/ESR, 2013. 251 p.

PFLEEGER, C. P; PFLEEGER, S. L. **Security in Computing**. 4. ed. Upper Saddle River, Nj, USA: Prentice Hall, 2007. 845 p.

SHIREY, R. **RFC 2828: Internet Security Glossary**. 2000. Disponível em: <<https://www.ietf.org/rfc/rfc2828.txt>>. Acesso em: 18 ago. 2017.

STALLINGS, William. **Criptografia e segurança de redes: Princípios e práticas**. 4. ed. São Paulo: Pearson Prentice Hall, 2008. 481 p.

STUTTARD, D; PINTO, M. **The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws**. 2. ed. Indianapolis, IN, USA: John Wiley & Sons, 2011. 878 p.

UTO, Nelson. **Teste de invasão de aplicações web**. Rio de Janeiro: RNP/ESR, 2013. 510 p.

UTO, N; MELO, S.P. Vulnerabilidades em Aplicações Web e Mecanismos de Proteção. Minicursos SBSeg 2009. **IX Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais**, Campinas, São Paulo, Brasil, 2009.

WEIDMAN, Georgia. **Testes de Invasão: Uma introdução prática ao hacking**. São Paulo: Novatec, 2014.

WILLIS, Chuck. **OWASP Broken Web Applications Project**. Disponível em: <https://www.owasp.org/index.php/OWASP_Broken_Web_Applications_Project>. Acesso em: 30 ago. 2017.

APÊNDICE A – ARTIGO

Guia de testes de segurança para aplicações web

Antonio Marco da Costa Taha¹

¹Departamento de Informática e Estatística – Universidade Federal de Santa Catarina (UFSC)
Caixa Postal 476 – 88.040-900 – Florianópolis – SC – Brasil
antoniotahasc@gmail.com

Abstract. *Web applications are increasingly frequent in people's daily lives and process sensitive data provided by their users, therefore they need to take care of the security of data. This paper aims to prepare a web application security testing guide, based on the OWASP methodology. The tests covered by the guide present through practical examples how to detect and exploit vulnerabilities in web systems and use security tools in their process. With the help of this guide, through a simple and intuitive script, undergraduate students, professionals with basic knowledge of system security and interested, can learn how to detect vulnerabilities in their applications and how they occur.*

Resumo. *As aplicações web são cada vez mais frequentes no cotidiano das pessoas e processam dados sensíveis fornecidos por seus usuários, portanto precisam cuidar da segurança desses dados. Este trabalho tem por objetivo elaborar um guia de testes de segurança para aplicações web, baseado na metodologia da OWASP. Os testes abordados pelo guia apresentam através de exemplos práticos como detectar e explorar vulnerabilidades em sistemas web e utilizam ferramentas de segurança no seu processo. Com o auxílio deste guia, através de um roteiro simples e intuitivo, alunos de graduação, profissionais com pouco conhecimento em segurança de sistemas e interessados, podem aprender como detectar vulnerabilidades em suas aplicações e como elas surgem.*

1. Introdução

Há diversos sites e aplicações que fornecem serviços a usuários na web. Hoje existem serviços de *streaming* através dos quais usuários podem assistir a filmes e séries, seja no computador, dispositivo móvel ou televisão. Existem aplicações que permitem a troca de mensagens, a realização de conversas por áudio e vídeo. Sistemas de Internet banking, que são sistemas de acesso a contas bancárias através da Internet, permitindo a realização de operações financeiras por usuários autorizados. E o que todas essas aplicações possuem em comum?

Geralmente esses serviços solicitam aos seus usuários a realização de um cadastro de suas informações pessoais. Alguns cobram pelo acesso aos seus recursos e outros dizem garantir o sigilo de conversas através do uso de criptografia ponto-a-ponto. Como os

desenvolvedores dessas e das inúmeras aplicações que existem na web podem garantir a confiabilidade de seus sistemas?

Segundo Peixinho e Fonseca (2013), os desenvolvedores de software possuem cada vez menos tempo para a entrega do produto final ao cliente, que tem se tornado cada vez mais exigente, e não possuem o conhecimento técnico necessário para garantir a segurança de uma aplicação.

Stallings (2008) comenta que devido ao aumento na complexidade das aplicações e do acesso à Internet, bem como o uso de recursos que dependem cada vez mais de conexão com a rede, é necessário cada vez mais profissionais com conhecimento técnico em segurança e tecnologias capazes de agir contra as ameaças aos sistemas.

Existem padrões de projetos e técnicas de programação que permitem aos desenvolvedores produzirem código mais seguro, evitando vulnerabilidades nas aplicações conectadas à Internet.

Com o objetivo de detectar tais vulnerabilidades nos sistemas, impedir o acesso não autorizado de pessoas a dados sigilosos e que estes dados não sejam adulterados, além de evitar indisponibilidade das aplicações devido a falhas de segurança, devem ser realizadas auditorias de segurança. Peixinho e Fonseca (2013) definem que auditoria trata-se de um processo de comparação de algo contra um padrão estabelecido e destacam que em sistemas de informação, vários parâmetros podem ser utilizados para auditar um sistema, como a política de segurança organizacional ou se um sistema está atualizado conforme as novas correções de segurança disponíveis no mercado.

Este trabalho tem como objetivo a elaboração de um guia de testes de intrusão voltado às aplicações web, tendo como base a metodologia proposta pela Open Web Application Security Project (OWASP) em seu documento Testing Guide, atualmente na versão 4.0 (OWASP, 2014).

2. Fundamentação Teórica

2.1 Confidencialidade, Integridade e Disponibilidade

A confidencialidade, a integridade e a disponibilidade (CID), compõem as três propriedades fundamentais de segurança, que são conhecidos como seus três pilares de sustentação.

Recursos computacionais, documentos e informações que devem estar protegidos e em segredo só podem ser acessados por entidades com autorização para isso. Segundo Pfleeger e Pfleeger (2007), a confidencialidade é a propriedade que garante que esses recursos só possam ser acessados por quem realmente tem autorização, e se não a tiver, não deve ser possível visualizar, acessar, distribuir ou imprimir algo.

Pode-se citar como exemplo os dados de um prontuário médico eletrônico. Esses dados somente podem ser visualizados pelos profissionais da saúde que tenham permissão e acesso ao prontuário e pelo próprio paciente a quem se destina o prontuário, e caso alguém não autorizado tenha acesso a esse documento, inclusive físico, perde-se a propriedade de confidencialidade.

A propriedade de integridade tem como objetivo garantir se dada informação ou recurso é confiável ou não, normalmente trata da integridade dos dados, ou seja, do conteúdo da informação, e da origem desses dados (BISHOP, 2005). Uma informação é íntegra e confiável se ela não sofreu nenhuma adulteração e se for possível confirmar ou houver garantias da integridade da sua origem. Pode-se citar como exemplo o caso de um usuário

realizando o *download* de uma imagem de disco no formato .iso e que ele gostaria de saber se os dados estão íntegros, que não tenha ocorrido nenhuma falha durante o processo, sem perder nenhum dado, ou que um software mal-intencionado instalado em seu computador não tenha modificado os dados, ou até mesmo ter realizado o *download* de um arquivo que é dito ser de tal origem, mas na realidade é de outra.

Em Bishop (2005) são apresentadas duas formas de garantia dessa propriedade. A primeira forma trata dos mecanismos de prevenção, que visam bloquear tentativas não autorizadas de mudar os dados e o uso de caminhos não autorizados para mudar os dados. A segunda forma são mecanismos de detecção que informam que a integridade dos dados não é mais confiável, podendo usar recursos do sistema para gerar alertas com informações da causa da corrupção dos dados.

Segundo a Shirey (2000, p. 17) disponibilidade é a “propriedade de um sistema ou um recurso do sistema ser acessível e utilizável mediante demanda por uma entidade do sistema autorizada, de acordo com as especificações de desempenho do sistema.” Quando há disponibilidade, usuários ou serviços autorizados devem conseguir acessar dados, serviços e sistemas (PFLEEGER; PFLEEGER, 2007). Por exemplo, se um usuário paga por um serviço de *streaming de dados*, como serviços de transmissão de vídeos, filmes e séries, e no contrato com o sistema que promove o *streaming* é descrito que este deve estar disponível em dia e hora específico, o serviço deveria estar. Existem inúmeros problemas que podem tornar esses recursos indisponíveis, como desastres naturais, problemas de conexão de servidor, erros de desenvolvimento dos sistemas e indisponibilidade por quantidade de acessos maior que a suportada.

Um dos ataques mais conhecidos atualmente é o *Distributed Denial of Service* (DDOS), ataque de negação de serviço, que tem por objetivo tornar um sistema indisponível por meio da realização de acessos e envio de pacotes em larga escala à sistemas, com o intuito de ultrapassar a capacidade de processamento dos servidores desses sistemas, tornando-os lentos e impossíveis de se utilizar durante o ataque, ou até mesmo derrubando o serviço.

2.2 Vulnerabilidade e Ameaça

Para planejar a segurança de aplicações web, estabelecer medidas de prevenção, proteção à ataques e em realizar testes de intrusão para verificar a segurança de um sistema, é necessário saber quais vulnerabilidades e ameaças estão presentes.

Conforme OWASP (2014, p. 27) “uma vulnerabilidade é uma falha ou fraqueza no projeto, implementação, operação ou gerenciamento de um sistema que poderia ser explorada para comprometer os objetivos de segurança do sistema”. Atualmente são desenvolvidos softwares cada vez mais complexos, com mais requisitos de negócio e com milhares de linhas de código, e combinando isso com o fato dos desenvolvedores não terem muito conhecimento em segurança ou não terem tempo hábil para planejá-la, acabam gerando cada vez mais problemas em seus sistemas, que podem tornar-se vulnerabilidades de segurança (UTO; MELO, 2009).

“Uma ameaça é qualquer coisa (um atacante externo malicioso, um usuário interno, uma instabilidade do sistema) que pode prejudicar os ativos de uma aplicação (como os dados num banco de dados), explorando uma vulnerabilidade”, conforme OWASP (2014, p. 27).

Exemplificando num cenário, pode-se ter uma aplicação web que apresenta uma vulnerabilidade não corrigida e gerada devido ao desenvolvimento não ter adotado um padrão

de projeto para tratamento dos dados vindos através da requisição do cliente ao servidor, que permite que sejam obtidos dados sigilosos do banco de dados desse sistema. Como ameaça temos um agente mal-intencionado que caso descubra essa vulnerabilidade, pode realizar um ataque e obter os dados confidenciais.

2.3 OPEN WEB APPLICATION SECURITY PROJECT

A OWASP (*Open Web Application Security Project*) é uma comunidade mundial livre e aberta cujo objetivo é promover o conhecimento a respeito de segurança de software. No documento da OWASP (2014) é comentado que sua missão é tornar visível informações a respeito da segurança de aplicações, para que pessoas e organizações possam tomar decisões e analisar os riscos de segurança que seus sistemas possam enfrentar. Por se tratar de uma comunidade aberta, todos são livres para participar dos projetos que estão disponíveis no próprio site da comunidade, que segue o formato de wiki.

O trabalho da comunidade recebe o suporte da fundação OWASP, uma organização sem fins lucrativos estabelecida nos Estados Unidos em 2004. Como não é filiada a nenhuma organização ou empresa de tecnologia e não enfrenta pressões comerciais, consegue promover a divulgação de informações imparciais sobre a segurança de aplicações web.

Para o desenvolvimento deste trabalho é utilizada como base a metodologia de testes de intrusão de um dos projetos da OWASP, o *OWASP Testing Guide v4*. Nesta seção são explicadas algumas das técnicas de teste de segurança propostas pela metodologia OWASP, enfatizando testes de intrusão que é a técnica adotada neste trabalho e explicando quais tipos de teste foram selecionados para compor o guia desenvolvido.

2.3.1 Testes de Intrusão

Técnica onde testers de segurança buscam com o auxílio de ferramentas de automatização, vulnerabilidades de segurança numa aplicação que esteja em execução, sem precisar conhecer seu código fonte.

Segundo a Owasp (2014, p. 14), “os testers atuam como atacantes e tentam encontrar e explorar vulnerabilidades”. Weidman (2014, p. 30) afirma que “Testes de invasão ou *pentesting* envolvem a simulação de ataques reais para avaliar os riscos associados a potenciais brechas de segurança”, também comenta que os testers exploram as vulnerabilidades encontradas para verificar quais recursos poderiam ser obtidos pelos atacantes ao explorar esses problemas.

Durante o processo de *pentest*, o profissional que o estiver realizando deve elaborar um relatório contendo as vulnerabilidades encontradas, para que os desenvolvedores possam adotar as correções necessárias na aplicação web.

Conforme a Owasp (2014), existem vantagens ao realizar um teste de invasão, como o tempo menor para executá-lo, graças ao auxílio das ferramentas automatizadas de teste, nível de conhecimento menor em segurança pelo profissional que está executando o teste. Como desvantagem, os testes só podem ser realizados quando o estado do software já está num nível avançado do ciclo de desenvolvimento, já que é necessário que a aplicação esteja rodando para que o *pentest* possa ser executado.

2.3.2 Tipos de Teste

Nesta seção são definidos os dois tipos de teste da metodologia OWASP que são cobertos por este guia, com o intuito de auxiliar numa maior compreensão dos testes que podem ser realizados por quem utilizar este trabalho.

2.3.2.1 Teste de Autenticação

Autenticação é um processo que tem por objetivo provar que algo é verdadeiro ou confirmar a identidade de alguém. “Na segurança computacional, autenticação é o processo de tentar verificar a identidade digital de alguém que deseja se comunicar” (OWASP, 2014, p. 66).

O ato de um usuário tentar efetuar *login* em um sistema por meio de nome do usuário e senha é um processo de autenticação, no qual a aplicação em questão verificará se os dados informados são válidos e fornecerá ou não acesso aos seus recursos.

Mecanismos como assinatura e certificado digital, permitem que mensagens e documentos, enviados seja por e-mail ou por mecanismo de *upload* em aplicações web, possam ter sua autenticidade verificada. Os mecanismos também permitem o estabelecimento de comunicação entre o cliente de uma aplicação e seu servidor, por meio de protocolos criptográficos seguros.

Testes de autenticação focam seus esforços basicamente em verificar o funcionamento desses mecanismos, tentando obter algum tipo de informação, com o intuito de que seja possível efetuar uma autenticação falsa (OWASP, 2014).

2.3.2.2 Teste de Validação de Entrada

Segundo OWASP (2014), a maioria das vulnerabilidades principais de aplicações web são provenientes da validação inadequada de entrada dos dados vindos do cliente ou serviço que se comunica com a aplicação. Vulnerabilidades como *SQL injection*, *cross-site scripting* e *buffer overflows* são exemplos de problemas de segurança que acontecem devido a forma como os dados de uma aplicação estão sendo tratados.

Aplicações web muito grandes apresentam vários formulários com campos para entrada de dados. Pode-se citar como exemplo uma aplicação voltada para a área da saúde e nela teríamos a possibilidade de cadastrar dados de profissionais ou pacientes, e somente nesses dois cadastros já existiriam inúmeros campos que precisam de uma série de validações, para então poder enviar os dados ao servidor para serem salvos numa base de dados.

O teste de validação de entrada é realizado por meio de um conjunto de ações que visam verificar se as validações realizadas, em cima das entradas de dados da aplicação web, são suficientes, conforme OWASP (2014, p. 98).

2.4 Segurança de Aplicações Web

Neste capítulo serão descritas as principais vulnerabilidades presentes no documento da OWASP Top 10 que serão objeto de estudo nos cenários controlados presentes neste

trabalho. As vulnerabilidades foram escolhidas devido a sua prevalência nas aplicações web (OWASP, 2013).

2.4.1 Injeção de Código

OWASP (2013) afirma que as falhas de Injeção, tais como injeção de comandos SQL (*Structured Query Language*), ocorrem quando um atacante envia dados manipulados através de comandos ou consultas a um interpretador responsável por executá-los, tentando executar comandos ou obter controle de acesso quando ocorre o processamento desses dados.

Pode-se por exemplo, citar os casos de uso da técnica de *Sql Injection* e que sua principal causa é a concatenação de dados fornecidos por um usuário numa *string* que armazena um comando SQL a ser executado por um sistema gerenciador de banco de dados (OWASP, 2014). Para evitar essa injeção bastaria tratar essas *strings*, evitando o uso de caracteres especiais, ou adotando tecnologias mais recentes e seguras para construção desses comandos.

2.4.2 Quebra de autenticação e Gerenciamento de Sessão

As aplicações web apresentam muitos recursos disponíveis e alguns só podem ser acessados por categorias específicas de usuários, também em muitos casos, apresentam a necessidade de autenticação com o sistema para que suas funcionalidades sejam habilitadas.

Pode-se ter um sistema onde os usuários apresentam perfis e cada um deles tem um conjunto de permissões para a execução e acesso a módulos ou simplesmente estabelecer uma conexão entre cliente e servidor com determinado tempo de validade e ainda de maneira mais básica dar acesso a somente usuários cadastrados por meio de *login* e senha. Esses são exemplos básicos dos mecanismos de autenticação que podem ser utilizados.

Como requisitos de segurança no acesso às aplicações são utilizados em conjunto os mecanismos de autenticação, gerenciamento de sessão e controle de acesso (STUTTARD; PINTO, 2011). O resultado de uma autenticação realizada com sucesso é um conjunto de credenciais que estabelecem a sessão de um usuário. Essas credenciais podem ser *tokens* ou *cookies*, que são valores gerados pelo servidor da aplicação utilizando algoritmos criptográficos (OWASP, 2014).

2.4.3 Cross-Site Scripting

Trata-se de uma vulnerabilidade também conhecida por XSS, que permite a um atacante injetar código ou informações e executar scripts no navegador do cliente da aplicação. Geralmente ocorre devido à falta de validações ou filtros dos dados que o servidor recebe e que são usados para gerar páginas dinâmicas. Como o código injetado nessas páginas dinâmicas são válidos e interpretados por um navegador, o cliente que acessar a página sofrerá com a execução do script que foi injetado, podendo ter sua sessão e dados comprometidos (UTO; MELO, 2009).

Existem alguns tipos de vulnerabilidades de *Cross-Site Scripting*, as que serão abordadas neste trabalho são XSS Persistido e XSS Refletido:

XSS Persistido: Segundo a OWASP (2017) esse é considerado o tipo mais perigoso, visto que os dados inseridos pelo atacante na aplicação geralmente são armazenados no banco de dados e muitos usuários podem recuperar esses dados ao usar a aplicação, obtendo na realidade um código malicioso que será executado em seu navegador.

XSS Refletido: Esse tipo de XSS ocorre quando a entrada do usuário é retornada pela aplicação como resposta imediata após a realização de uma requisição ao servidor pelo usuário, na qual os dados faziam parte da própria requisição, conforme OWASP (2017).

3 Ferramentas

3.1 OWASP BROKEN WEB APPLICATIONS PROJECT

Conforme Willis (2017), trata-se de um projeto desenvolvido pela comunidade da OWASP, cujo objetivo é fornecer uma máquina virtual com uma coleção de aplicações web *open source* e vulneráveis, para que interessados na área de segurança possam realizar testes de intrusão seguros, permitindo na prática aprenderem mais e confirmarem a teoria.

Utilizando a máquina virtual OWASP Broken é possível aprender mais sobre segurança em aplicações web, realizar técnicas de testes manuais, utilizar ferramentas automatizadas de detecção de vulnerabilidades, testar ferramentas de análise de código e efetuar ataques controlados. Para a realização dos testes presentes neste guia, serão utilizadas aplicações vulneráveis que estão nesse projeto.

3.2 KALI LINUX

Segundo a *Offensive Security* (2017), trata-se de uma distribuição Linux baseada em Debian, que apresenta centenas de ferramentas especificamente para a realização de testes de penetração e auditoria de segurança. Através delas é possível estudar e realizar forense computacional, testes de intrusão e aprender a respeito de segurança. Durante a execução dos testes do guia serão utilizadas ferramentas presentes na Kali Linux para a detecção de vulnerabilidades em aplicações.

Esse sistema operacional, apresenta ferramentas para análise de vulnerabilidades, *scan* de portas de servidores, detecção de vulnerabilidades em aplicações web, ataques às senhas como força bruta, ferramentas de *sniffer* para monitorar e analisar o tráfego de rede, injeção de código malicioso, entre outras, contando com mais de 600 ferramentas de teste de intrusão.

3.3 OWASP ZED ATTACK PROXY

Mais conhecida como OWASP Zap, ver figura 1, é uma ferramenta de segurança automatizada mantida por centenas de voluntários e sua principal funcionalidade é realizar *scan* em aplicações web em busca de vulnerabilidades. Bennetts (2017) comenta que pode ser usada tanto por especialistas em segurança como desenvolvedores ou testers que estejam começando na área de testes de intrusão. Ela é uma ferramenta *open source*, fácil de usar e instalar, multi plataforma e traduzida em diversos idiomas.

Devido a sua facilidade de uso, essa ferramenta é adotada nos testes realizados neste guia, permitindo a identificação de vulnerabilidades nas aplicações e a partir disso a realização dos ataques. Ela pode ser usada tanto numa aplicação em produção como em fase de testes.

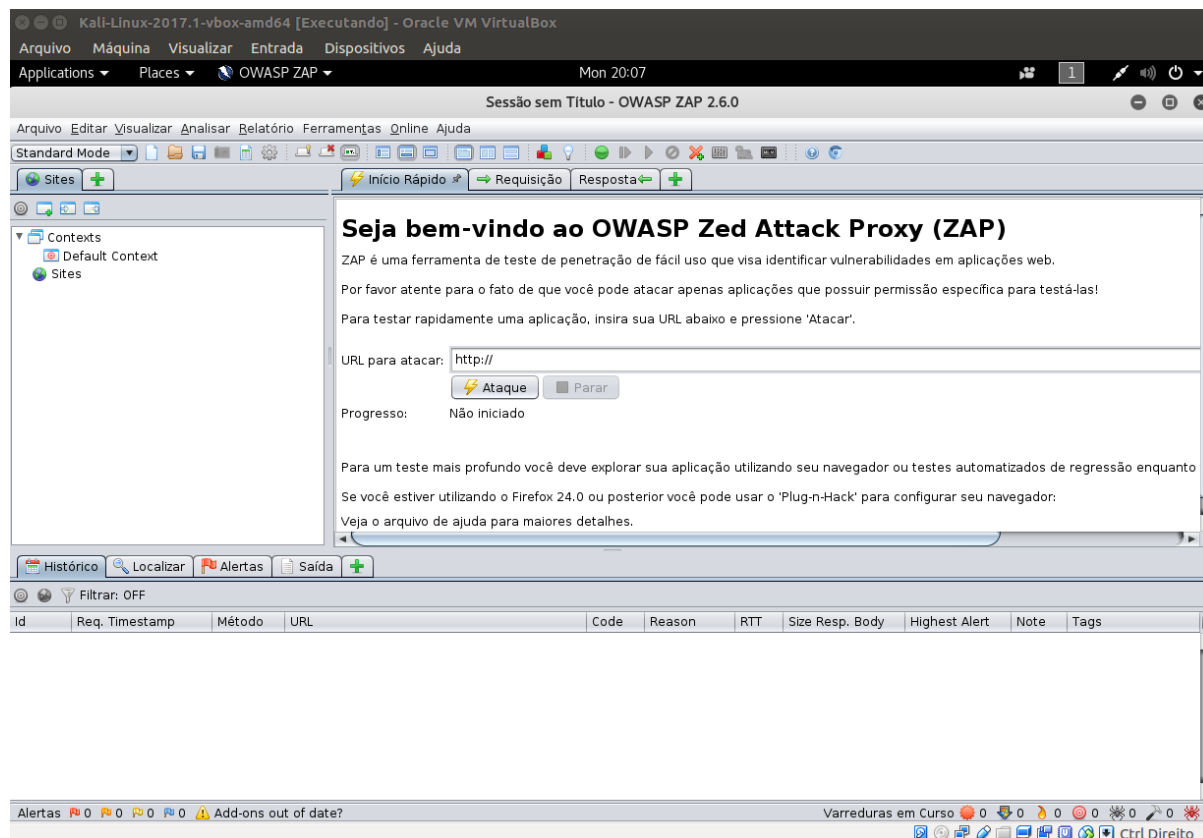


Figura 1 - OWASP Zap

4 Guia para Testes de Intrusão

4.1 Teste de *Cross-Site Scripting* Refletido

Para demonstrar o teste de *cross-site scripting* refletido foi escolhida a aplicação WordPress em sua versão 2.0.0, presente na máquina da Owasp Broken. WordPress é uma plataforma para publicação de conteúdo na Internet, mundialmente conhecida e que em versões antigas já apresentou vulnerabilidades.

Para acessar a aplicação WordPress, é necessário abrir o navegador Firefox, presente na máquina Kali. No *browser* ao acessar o endereço em que a máquina Owasp Broken foi configurada todas as aplicações presentes na máquina Owasp Broken estão disponíveis.

Na aplicação WordPress existe um formulário de registro, conforme figura 2. Ao executar um *scan* ativo na ferramenta Owasp Zap é possível detectar que os campos *Username* e *E-mail* estão vulneráveis a *cross-site scripting* refletido.



The image shows the WordPress registration form. At the top is the WordPress logo and the text "Register for this Blog". Below this are two input fields: "Username:" and "E-mail:". Under the "E-mail:" field, it says "A password will be emailed to you." At the bottom right is a "Register »" button. At the bottom left are three links: « Back to blog, Login, and Lost your password?

Figura 02 - Página de registro para teste

Na tela da figura 3 uma caixa de alerta é exibida ao tentar registrar como *E-mail* a *string* de ataque valor "><script>alert(`Cross-site scripting refletido`);</script>".

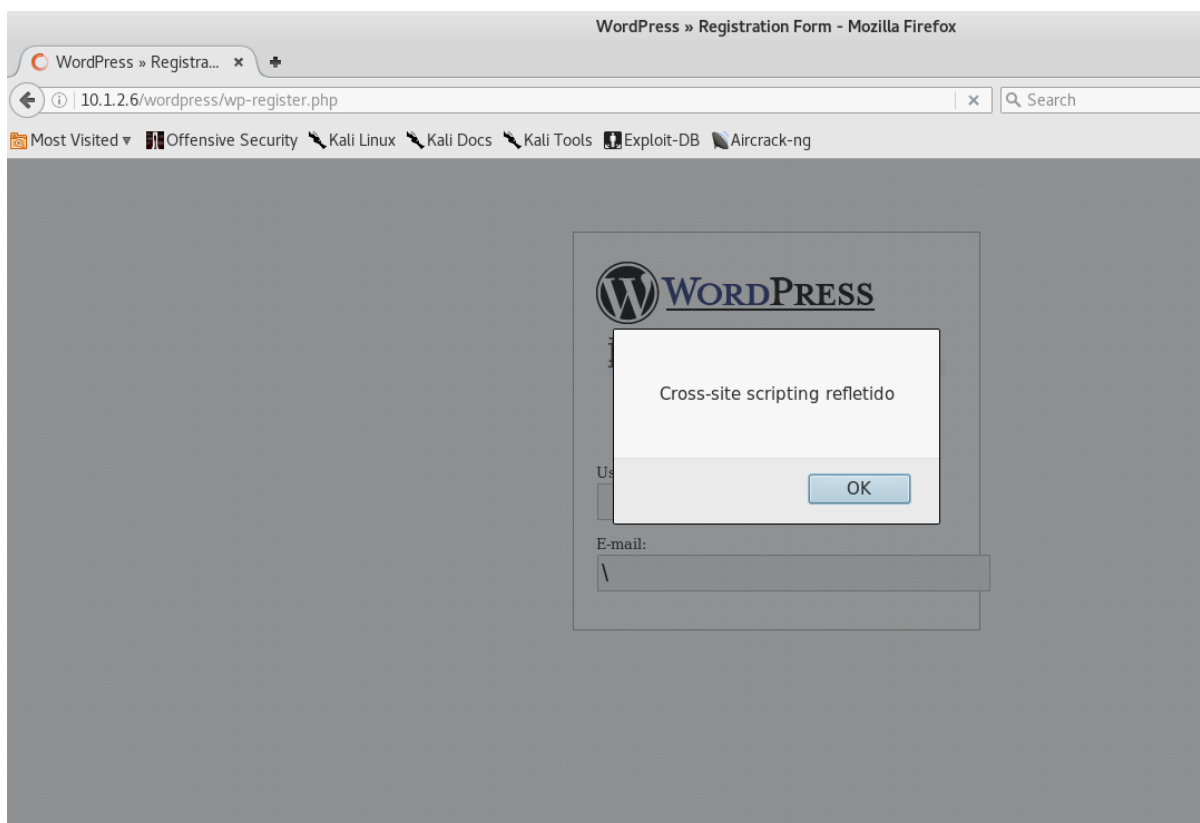


Figura 03 - Exploração de vulnerabilidade de XSS Refletido

4.2 Teste de *Cross-Site Scripting* Persistido

Para realizar o teste de *cross-site scripting* persistido foi escolhida a aplicação Mutillidae II, encontrada na Owasp Broken. Essa aplicação fornece um ambiente onde é possível treinar e explorar diversos tipos de vulnerabilidades web.

Na Mutillidae foi escolhido para realizar o teste um formulário de comentários de um blog, conforme figura 4. Ao executar uma varredura ativa nos campos do formulário é possível perceber que o campo de comentário está vulnerável a *cross-site scripting* persistido.

Add blog for anonymous

Note: ,<i> and <u> are now allowed in blog entries

`</td><script>alert("Cross-site scripting persistido");</script><td>`

Save Blog Entry

Figura 04 - Salvar código malicioso como comentário

Ao inserir um dado malicioso no formato `</td><script>alert("Cross-site scripting persistido");</script><td>` no campo de comentário e salvar, a aplicação salva um código malicioso que será carregado no navegador do usuário que acessar a página vulnerável e exibirá um alerta com a mensagem “*Cross-site scripting persistido*”, assim como na figura 5.

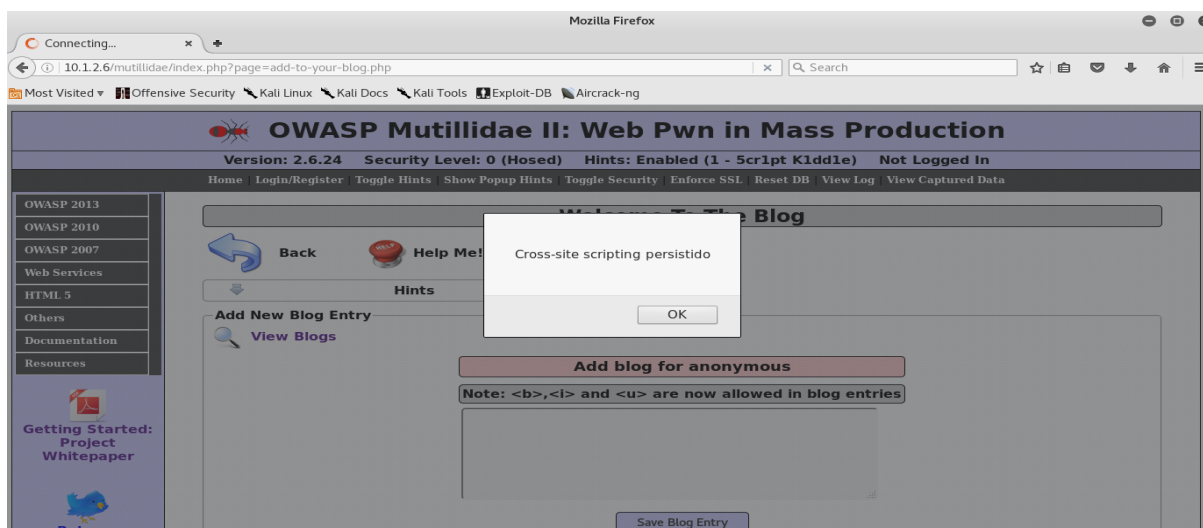


Figura 05 - Exploração de vulnerabilidade de XSS Persistido

4.3 Teste de SQL Injection

Para o teste deve-se abrir a aplicação Mutillidae. A página que será testada apresenta um formulário, figura 6, que dado um *name* e *password* retorna informações a respeito de um usuário salvo na base de dados da aplicação.

Figura 06 - Formulário do teste de injeção SQL

Na ferramenta Owasp Zap ao fazer uma varredura ativa é possível ver que foram encontradas duas vulnerabilidades de injeção de SQL, uma em cada campo da tela. Ao digitar como valor do campo *Name* a *string* de ataque ' or 1=1 # pode-se recuperar na tela da aplicação todos os usuários cadastrados na base, conforme figura 7.

```

Results for "' or 1=1 #".25 records found.

Username=admin
Password=admin
Signature=g0t r00t?

Username=adrian
Password=somepassword
Signature=Zombie Films Rock!

Username=john
Password=monkey
Signature=I like the smell of confunk

Username=jeremy
Password=password
Signature=d1373 1337 speak

Username=bryce
Password=password
Signature=I Love SANS

Username=samurai
Password=samurai
Signature=Carving fools

Username=jim
Password=password
Signature=Rome is burning

Username=bobby
Password=password
Signature=Hank is my dad

Username=simba
Password=password
Signature=I am a super-cat

Username=drevell
Password=password
Signature=Preparation H

```

Figura 07 - Resultado da injeção de código SQL

4.4 Teste para burlar esquema de autenticação com injeção SQL

Para realizar esse teste foi escolhida uma aplicação web conhecida por WackoPicko. Ela é uma aplicação desenvolvida para testes de segurança, simulando uma aplicação real de um sistema de *upload* de imagens, contendo autenticação de usuários e comentários.

A WackoPicko apresenta um formulário para autenticação dos usuários que desejam realizar o upload de imagens, figura 8. Ao realizar um *scan* ativo com a ferramenta Owasp Zap foi possível detectar vulnerabilidade de injeção SQL no campo *Username*.



Figura 08 - Formulário de login WackoPicko

A *string* de ataque ' or 1=1 # é criada e inserida no campo de *Username*. Ao realizar a requisição de *login*, o esquema de autenticação deve ter sido burlado e um determinado usuário retornado pela consulta SQL da aplicação estará autenticado no sistema, conforme figura 9.

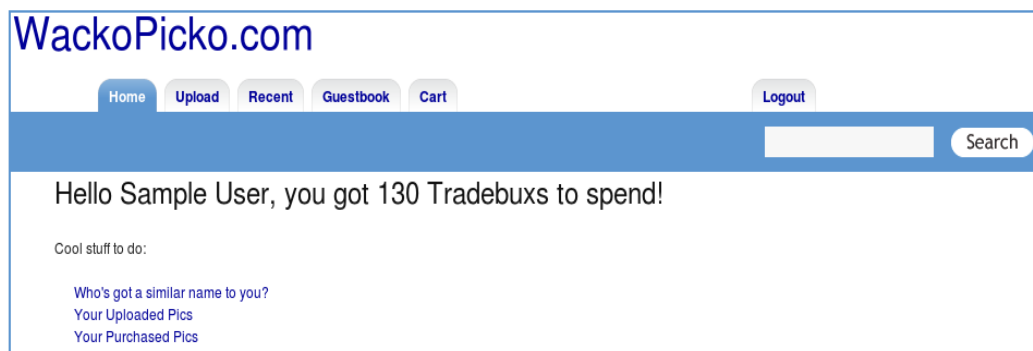


Figura 09 - Esquema de autenticação burlado por injeção SQL

4.5 Teste para burlar esquema de autenticação via *cookie*

Para executar o teste deve-se registrar um usuário e se autenticar com ele na aplicação Mutillidae. Após a autenticação pode-se observar através de inspeção dos cookies do

navegador, figura 10, que o usuário apresenta um cookie uid que representa o usuário autenticado na aplicação.

Method	File	Domain	Headers	Cookies	Params
200	GET	index.php?popUpNotificationCode...	10.1.2.6		
200	GET	global-styles.css	10.1.2.6		
200	GET	ddsmoothmenu.css	10.1.2.6		
200	GET	ddsmoothmenu-v.css	10.1.2.6		
200	GET	bookmark-site.js	10.1.2.6		
200	GET	ddsmoothmenu.js	10.1.2.6		
200	GET	jquery.min.js	10.1.2.6		
200	GET	jquery.js	10.1.2.6		
200	GET	jquery.balloon.js	10.1.2.6		

Filter cookies
Request cookies
PHPSESSID: "dndb94k5i40up3u7jmi"
acgroupswithpersist: "nada"
acopendivids: "swingset,jotto,phpbt"
jiveLastVisited: "1508765577696"
showhints: "1"
tz_offset: "-14400"
uid: "25"
username: "bobtester"

Figura 10 - Cookies da aplicação Mutillidae

Se o uid do usuário for alterado nos cookies, pode-se notar que a aplicação utiliza esse *token* como forma de verificar qual usuário está autenticado e que recursos do sistema ele terá acesso. Alterando o valor do uid para 1, o usuário admin estará autenticado no sistema, conforme figura 11, e um atacante teria acesso aos seus recursos.



Figura 11 - Usuário administrador autenticado

5 Conclusão

Neste trabalho foi desenvolvido um guia para realização de testes de segurança em aplicações web, destinado aos profissionais que trabalham com desenvolvimento de software, interessados pela área e alunos de graduação.

O guia aborda a detecção das vulnerabilidades de *cross-site scripting* persistido e refletido, de injeção de código e de autenticação, descritas no documento criado pela OWASP das 10 vulnerabilidades mais comuns em aplicações web. Para cobrir esses problemas de segurança foram realizados testes de autenticação e de validação de entrada de dados, selecionados a partir da metodologia da OWASP, em cima de aplicações desenvolvidas para testes de intrusão e sistemas reais em versões antigas e vulneráveis.

Com o auxílio deste guia, profissionais com pouco conhecimento em segurança de sistemas são capazes de testar suas aplicações e aprender como as vulnerabilidades abordadas surgem, como detectá-las através de ferramentas de varredura e como evitar.

Como trabalhos futuros é possível explorar os demais tipos de teste presentes na metodologia da OWASP complementando este guia, aplicar os testes em sistemas reais,

elaborar guias para outras técnicas de testes como modelagem de ameaças e revisão de código fonte.

Referências

BENNETTS, Simon. **OWASP Zed Attack Proxy Project**. Disponível em: <https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project>. Acesso em: 04 set. 2017.

BISHOP, Matt. **Introduction to Computer Security**. Boston, Ma: Addison-wesley, 2005. 784 p.

MEUCCI, Matteo; MULLER, Andrew. **OWASP Testing Guide v4**. Maryland, Eua: Owasp, 2014. 222 p.

OFFENSIVE SECURITY. **What is Kali Linux ?** Disponível em: <<https://docs.kali.org/introduction/what-is-kali-linux>>. Acesso em: 01 set. 2017.
OWASP. **OWASP Top 10**: Os dez riscos de segurança mais críticos em aplicações web. São Paulo: Owasp, 2013. 23 p

PEIXINHO, Ivo de Carvalho; FONSECA, Francisco Marmo da; LIMA, Francisco Marcelo Marques. **Segurança de Redes e Sistemas**. Rio de Janeiro: RNP/ESR, 2013. 251 p.

PFLEEGER, C. P; PFLEEGER, S. L. **Security in Computing**. 4. ed. Upper Saddle River, Nj, USA: Prentice Hall, 2007. 845 p.

SHIREY, R. **RFC 2828**: Internet Security Glossary. 2000. Disponível em: <<https://www.ietf.org/rfc/rfc2828.txt>>. Acesso em: 18 ago. 2017.

STALLINGS, William. **Criptografia e segurança de redes**: Princípios e práticas. 4. ed. São Paulo: Pearson Prentice Hall, 2008. 481 p.

STUTTARD, D; PINTO, M. **The Web Application Hacker's Handbook**: Finding and Exploiting Security Flaws. 2. ed. Indianapolis, IN, USA: John Wiley & Sons, 2011. 878 p.
UTO, N; MELO, S.P. Vulnerabilidades em Aplicações Web e Mecanismos de Proteção. Minicursos SBSeg 2009. **IX Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais**, Campinas, São Paulo, Brasil, 2009.

WEIDMAN, Georgia. **Testes de Invasão**: Uma introdução prática ao hacking. São Paulo: Novatec, 2014.

WILLIS, Chuck. **OWASP Broken Web Applications Project**. Disponível em: <https://www.owasp.org/index.php/OWASP_Broken_Web_Applications_Project>. Acesso em: 30 ago. 2017.